CSimpl

David Sanan, Yongwang Zhao

January 12, 2020

Contents

1 The Simpl Syntax

theory *Language* **imports** *HOL−Library.Old-Recdef* **begin**

1.1 The Core Language

We use a shallow embedding of boolean expressions as well as assertions as sets of states.

type-synonym $'s\ bexp = 's\ set$
type-synonym $'s\ assn = 's\ set$

datatype $(dead\ 's,\ 'p,\ 'f)\ com =$
$\quad Skip$
$\ |\ Basic\ 's \Rightarrow 's$
$\ |\ Spec\ ('s \times 's)\ set$
$\ |\ Seq\ ('s\ ,'p,\ 'f)\ com\ ('s,'p,\ 'f)\ com$
$\ |\ Cond\ 's\ bexp\ ('s,'p,'f)\ com\ \ ('s,'p,'f)\ com$
$\ |\ While\ 's\ bexp\ ('s,'p,'f)\ com$
$\ |\ Call\ 'p$
$\ |\ DynCom\ 's \Rightarrow ('s,'p,'f)\ com$
$\ |\ Guard\ 'f\ 's\ bexp\ ('s,'p,'f)\ com$
$\ |\ Throw$
$\ |\ Catch\ ('s,'p,'f)\ com\ ('s,'p,'f)\ com$

abbreviation (*input*)
$\quad set\text{-}fun :: 'a\ set \Rightarrow 'a \Rightarrow bool\ \ (\text{-}_f)$ **where**
$\quad set\text{-}fun\ s \equiv \lambda v.\ v{\in}s$

abbreviation (*input*)
$\quad fun\text{-}set :: ('a \Rightarrow bool) \Rightarrow 'a\ set\ \ (\text{-}_s)$ **where**
$\quad fun\text{-}set\ f \equiv \{\sigma.\ f\ \sigma\}$

1.2 Derived Language Constructs

definition
$\quad raise{::}\ ('s \Rightarrow 's) \Rightarrow ('s,'p,'f)\ com$ **where**
$\quad raise\ f = Seq\ (Basic\ f)\ Throw$

definition
$\quad condCatch{::}\ ('s,'p,'f)\ com \Rightarrow 's\ bexp \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com$ **where**
$\quad condCatch\ c_1\ b\ c_2 = Catch\ c_1\ (Cond\ b\ c_2\ Throw)$

definition
$\quad bind{::}\ ('s \Rightarrow 'v) \Rightarrow ('v \Rightarrow ('s,'p,'f)\ com) \Rightarrow ('s,'p,'f)\ com$ **where**
$\quad bind\ e\ c = DynCom\ (\lambda s.\ c\ (e\ s))$

definition
$\quad bseq{::}\ ('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com$ **where**
$\quad bseq = Seq$

definition
$block{::}\ [{'s}{\Rightarrow}{'s},({'s},{'p},{'f})\ com,{'s}{\Rightarrow}{'s}{\Rightarrow}{'s},{'s}{\Rightarrow}{'s}{\Rightarrow}({'s},{'p},{'f})\ com]{\Rightarrow}({'s},{'p},{'f})\ com$
where
$block\ init\ bdy\ return\ c =$
$DynCom\ (\lambda s.\ (Seq\ (Catch\ (Seq\ (Basic\ init)\ bdy)\ (Seq\ (Basic\ (return\ s))\ Throw))$

$(DynCom\ (\lambda t.\ Seq\ (Basic\ (return\ s))\ (c\ s\ t))))$
$)$

definition
$call{::}\ ({'s}{\Rightarrow}{'s})\ \Rightarrow\ {'p}\ \Rightarrow\ ({'s}\ \Rightarrow\ {'s}\ \Rightarrow\ {'s}){\Rightarrow}({'s}{\Rightarrow}{'s}{\Rightarrow}({'s},{'p},{'f})\ com){\Rightarrow}({'s},{'p},{'f})com$
where
$call\ init\ p\ return\ c = block\ init\ (Call\ p)\ return\ c$

definition
$dynCall{::}\ ({'s}\ \Rightarrow\ {'s})\ \Rightarrow\ ({'s}\ \Rightarrow\ {'p})\ \Rightarrow$
$({'s}\ \Rightarrow\ {'s}\ \Rightarrow\ {'s})\ \Rightarrow\ ({'s}\ \Rightarrow\ {'s}\ \Rightarrow\ ({'s},{'p},{'f})\ com)\ \Rightarrow\ ({'s},{'p},{'f})\ com$ **where**
$dynCall\ init\ p\ return\ c = DynCom\ (\lambda s.\ call\ init\ (p\ s)\ return\ c)$

definition
$fcall{::}\ ({'s}{\Rightarrow}{'s})\ \Rightarrow\ {'p}\ \Rightarrow\ ({'s}\ \Rightarrow\ {'s}\ \Rightarrow\ {'s}){\Rightarrow}({'s}\ \Rightarrow\ {'v})\ \Rightarrow\ ({'v}{\Rightarrow}({'s},{'p},{'f})\ com)$
${\Rightarrow}({'s},{'p},{'f})com$ **where**
$fcall\ init\ p\ return\ result\ c = call\ init\ p\ return\ (\lambda s\ t.\ c\ (result\ t))$

definition
$lem{::}\ {'x}\ \Rightarrow\ ({'s},{'p},{'f})com\ {\Rightarrow}({'s},{'p},{'f})com$ **where**
$lem\ x\ c = c$

primrec $switch{::}\ ({'s}\ \Rightarrow\ {'v})\ \Rightarrow\ ({'v}\ set\ \times\ ({'s},{'p},{'f})\ com)\ list\ \Rightarrow\ ({'s},{'p},{'f})\ com$
where
$switch\ v\ []\ = Skip\ |$
$switch\ v\ (Vc\#vs) = Cond\ \{s.\ v\ s \in fst\ Vc\}\ (snd\ Vc)\ (switch\ v\ vs)$

definition $guaranteeStrip{::}\ {'f}\ \Rightarrow\ {'s}\ set \Rightarrow ({'s},{'p},{'f})\ com \Rightarrow ({'s},{'p},{'f})\ com$
where $guaranteeStrip\ f\ g\ c = Guard\ f\ g\ c$

definition $guaranteeStripPair{::}\ {'f}\ \Rightarrow\ {'s}\ set \Rightarrow ({'f}\ \times\ {'s}\ set)$
where $guaranteeStripPair\ f\ g = (f,g)$

primrec $guards{::}\ ({'f}\ \times\ {'s}\ set\ )\ list \Rightarrow ({'s},{'p},{'f})\ com \Rightarrow ({'s},{'p},{'f})\ com$
where
$guards\ []\ c = c\ |$
$guards\ (g\#gs)\ c = Guard\ (fst\ g)\ (snd\ g)\ (guards\ gs\ c)$

definition
$while{::}\ \ ({'f}\ \times\ {'s}\ set)\ list \Rightarrow {'s}\ bexp \Rightarrow ({'s},{'p},{'f})\ com \Rightarrow ({'s},\ {'p},\ {'f})\ com$
where
$while\ gs\ b\ c = guards\ gs\ (While\ b\ (Seq\ c\ (guards\ gs\ Skip)))$

definition
  whileAnno::
  $'s\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com$ **where**
  $whileAnno\ b\ I\ V\ c = While\ b\ c$

definition
  whileAnnoG::
  $('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow$
    $('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com$ **where**
  $whileAnnoG\ gs\ b\ I\ V\ c = while\ gs\ b\ c$

definition
  specAnno:: $('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s,'p,'f)\ com) \Rightarrow$
    $('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('s,'p,'f)\ com$
  where $specAnno\ P\ c\ Q\ A = (c\ undefined)$

definition
  whileAnnoFix::
  $'s\ bexp \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s \times 's)\ assn) \Rightarrow ('a \Rightarrow ('s,'p,'f)\ com) \Rightarrow$
    $('s,'p,'f)\ com$ **where**
  $whileAnnoFix\ b\ I\ V\ c = While\ b\ (c\ undefined)$

definition
  whileAnnoGFix::
  $('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s \times 's)\ assn) \Rightarrow$
    $('a \Rightarrow ('s,'p,'f)\ com) \Rightarrow ('s,'p,'f)\ com$ **where**
  $whileAnnoGFix\ gs\ b\ I\ V\ c = while\ gs\ b\ (c\ undefined)$

definition *if-rel*::$('s \Rightarrow bool) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \times 's)$ *set*
  where *if-rel* $b\ f\ g\ h = \{(s,t).\ if\ b\ s\ then\ t = f\ s\ else\ t = g\ s \vee t = h\ s\}$

lemma *fst-guaranteeStripPair*: *fst* (*guaranteeStripPair f g*) = *f*
  by (*simp add*: *guaranteeStripPair-def*)

lemma *snd-guaranteeStripPair*: *snd* (*guaranteeStripPair f g*) = *g*
  by (*simp add*: *guaranteeStripPair-def*)

1.3 Operations on Simpl-Syntax

1.3.1 Normalisation of Sequential Composition: *sequence*, *flatten* and *normalize*

primrec *flatten*:: $('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com\ list$
where
flatten Skip = [*Skip*] |
flatten (*Basic f*) = [*Basic f*] |
flatten (*Spec r*) = [*Spec r*] |
flatten (*Seq* c_1 c_2) = *flatten* c_1 @ *flatten* c_2 |

flatten (*Cond* b c_1 c_2) = [*Cond* b c_1 c_2] |
flatten (*While* b c) = [*While* b c] |
flatten (*Call* p) = [*Call* p] |
flatten (*DynCom* c) = [*DynCom* c] |
flatten (*Guard* f g c) = [*Guard* f g c] |
flatten *Throw* = [*Throw*] |
flatten (*Catch* c_1 c_2) = [*Catch* c_1 c_2]

primrec *sequence*:: ((′*s*,′*p*,′*f*) *com* ⇒ (′*s*,′*p*,′*f*) *com* ⇒ (′*s*,′*p*,′*f*) *com*) ⇒
(′*s*,′*p*,′*f*) *com list* ⇒ (′*s*,′*p*,′*f*) *com*
where
sequence seq [] = *Skip* |
sequence seq (*c*#*cs*) = (*case cs of* [] ⇒ *c*
| - ⇒ *seq c* (*sequence seq cs*))

primrec *normalize*:: (′*s*,′*p*,′*f*) *com* ⇒ (′*s*,′*p*,′*f*) *com*
where
normalize Skip = *Skip* |
normalize (*Basic f*) = *Basic f* |
normalize (*Spec r*) = *Spec r* |
normalize (*Seq* c_1 c_2) = *sequence Seq*
((*flatten* (*normalize* c_1)) @ (*flatten* (*normalize* c_2))) |
normalize (*Cond* b c_1 c_2) = *Cond* b (*normalize* c_1) (*normalize* c_2) |
normalize (*While* b c) = *While* b (*normalize* c) |
normalize (*Call* p) = *Call* p |
normalize (*DynCom* c) = *DynCom* (λ*s*. (*normalize* (*c s*))) |
normalize (*Guard* f g c) = *Guard* f g (*normalize* c) |
normalize Throw = *Throw* |
normalize (*Catch* c_1 c_2) = *Catch* (*normalize* c_1) (*normalize* c_2)

lemma *flatten-nonEmpty*: *flatten* $c \neq$ []
by (*induct c*) *simp-all*

lemma *flatten-single*: ∀ *c* ∈ *set* (*flatten* c'). *flatten* *c* = [*c*]
apply (*induct* c')
apply *simp*
apply *simp*
apply *simp*
apply (*simp* (*no-asm-use*))
apply *blast*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))

done

lemma *flatten-sequence-id*:
$[\![cs \neq []; \forall\, c \in set\ cs.\ flatten\ c = [c]]\!] \Longrightarrow flatten\ (sequence\ Seq\ cs) = cs$
apply (*induct cs*)
apply *simp*
apply (*case-tac cs*)
apply *simp*
apply *auto*
done

lemma *flatten-app*:
flatten (*sequence Seq* (*flatten c1* @ *flatten c2*)) = *flatten c1* @ *flatten c2*
apply (*rule flatten-sequence-id*)
apply (*simp add*: *flatten-nonEmpty*)
apply (*simp*)
apply (*insert flatten-single*)
apply *blast*
done

lemma *flatten-sequence-flatten*: *flatten* (*sequence Seq* (*flatten c*)) = *flatten c*
apply (*induct c*)
apply (*auto simp add*: *flatten-app*)
done

lemma *sequence-flatten-normalize*: *sequence Seq* (*flatten* (*normalize c*)) = *normalize c*
apply (*induct c*)
apply (*auto simp add*: *flatten-app*)
done

lemma *flatten-normalize*: $\bigwedge x\ xs.\ flatten\ (normalize\ c) = x\#xs$
$\Longrightarrow (case\ xs\ of\ [] \Rightarrow normalize\ c = x$
$\mid (x'\#xs') \Rightarrow normalize\ c = Seq\ x\ (sequence\ Seq\ xs))$
proof (*induct c*)
case (*Seq c1 c2*)
have *flatten* (*normalize* (*Seq c1 c2*)) = *x* # *xs* **by** *fact*
hence *flatten* (*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*))) =
x#xs
by *simp*
hence *x-xs*: *flatten* (*normalize c1*) @ *flatten* (*normalize c2*) = *x* # *xs*
by (*simp add*: *flatten-app*)
show *?case*

```
  proof (cases flatten (normalize c1))
    case Nil
    with flatten-nonEmpty show ?thesis by auto
  next
    case (Cons x1 xs1)
    note Cons-x1-xs1 = this
    with x-xs obtain
      x-x1: x=x1 and xs-rest: xs=xs1@flatten (normalize c2)
      by auto
    show ?thesis
    proof (cases xs1)
      case Nil
      from Seq.hyps (1) [OF Cons-x1-xs1] Nil
      have normalize c1 = x1
        by simp
      with Cons-x1-xs1 Nil x-x1 xs-rest show ?thesis
        apply (cases flatten (normalize c2))
        apply (fastforce simp add: flatten-nonEmpty)
        apply simp
        done
    next
      case Cons
      from Seq.hyps (1) [OF Cons-x1-xs1] Cons
      have normalize c1 = Seq x1 (sequence Seq xs1)
        by simp
      with Cons-x1-xs1 Nil x-x1 xs-rest show ?thesis
        apply (cases flatten (normalize c2))
        apply (fastforce simp add: flatten-nonEmpty)
        apply (simp split: list.splits)
        done
    qed
  qed
qed (auto)

lemma flatten-raise [simp]: flatten (raise f) = [Basic f, Throw]
  by (simp add: raise-def)

lemma flatten-condCatch [simp]: flatten (condCatch c1 b c2) = [condCatch c1 b
c2]
  by (simp add: condCatch-def)

lemma flatten-bind [simp]: flatten (bind e c) = [bind e c]
  by (simp add: bind-def)

lemma flatten-bseq [simp]: flatten (bseq c1 c2) = flatten c1 @ flatten c2
  by (simp add: bseq-def)

lemma flatten-block [simp]:
  flatten (block init bdy return result) = [block init bdy return result]
```

by (*simp add*: *block-def*)

lemma *flatten-call* [*simp*]: *flatten* (*call init p return result*) = [*call init p return result*]
by (*simp add*: *call-def*)

lemma *flatten-dynCall* [*simp*]: *flatten* (*dynCall init p return result*) = [*dynCall init p return result*]
by (*simp add*: *dynCall-def*)

lemma *flatten-fcall* [*simp*]: *flatten* (*fcall init p return result c*) = [*fcall init p return result c*]
by (*simp add*: *fcall-def*)

lemma *flatten-switch* [*simp*]: *flatten* (*switch v Vcs*) = [*switch v Vcs*]
by (*cases Vcs*) *auto*

lemma *flatten-guaranteeStrip* [*simp*]:
flatten (*guaranteeStrip f g c*) = [*guaranteeStrip f g c*]
by (*simp add*: *guaranteeStrip-def*)

lemma *flatten-while* [*simp*]: *flatten* (*while gs b c*) = [*while gs b c*]
apply (*simp add*: *while-def*)
apply (*induct gs*)
apply *auto*
done

lemma *flatten-whileAnno* [*simp*]:
flatten (*whileAnno b I V c*) = [*whileAnno b I V c*]
by (*simp add*: *whileAnno-def*)

lemma *flatten-whileAnnoG* [*simp*]:
flatten (*whileAnnoG gs b I V c*) = [*whileAnnoG gs b I V c*]
by (*simp add*: *whileAnnoG-def*)

lemma *flatten-specAnno* [*simp*]:
flatten (*specAnno P c Q A*) = *flatten* (*c undefined*)
by (*simp add*: *specAnno-def*)

lemmas *flatten-simps* = *flatten.simps flatten-raise flatten-condCatch flatten-bind flatten-block flatten-call flatten-dynCall flatten-fcall flatten-switch flatten-guaranteeStrip flatten-while flatten-whileAnno flatten-whileAnnoG flatten-specAnno*

lemma *normalize-raise* [*simp*]:
normalize (*raise f*) = *raise f*
by (*simp add*: *raise-def*)

lemma *normalize-condCatch* [*simp*]:

normalize (*condCatch c1 b c2*) = *condCatch* (*normalize c1*) *b* (*normalize c2*)
by (*simp add*: *condCatch-def*)

lemma *normalize-bind* [*simp*]:
normalize (*bind e c*) = *bind e* (λ*v*. *normalize* (*c v*))
by (*simp add*: *bind-def*)

lemma *normalize-bseq* [*simp*]:
normalize (*bseq c1 c2*) = *sequence bseq*
((*flatten* (*normalize c1*)) @ (*flatten* (*normalize c2*)))
by (*simp add*: *bseq-def*)

lemma *normalize-block* [*simp*]: *normalize* (*block init bdy return c*) =
block init (*normalize bdy*) *return* (λ*s t*. *normalize* (*c s t*))
apply (*simp add*: *block-def*)
apply (*rule ext*)
apply (*simp*)
apply (*cases flatten* (*normalize bdy*))
apply (*simp add*: *flatten-nonEmpty*)
apply (*rule conjI*)
apply *simp*
apply (*drule flatten-normalize*)
apply (*case-tac list*)
apply *simp*
apply *simp*
apply (*rule ext*)
apply (*case-tac flatten* (*normalize* (*c s sa*)))
apply (*simp add*: *flatten-nonEmpty*)
apply *simp*
apply (*thin-tac flatten* (*normalize bdy*) = *P* **for** *P*)
apply (*drule flatten-normalize*)
apply (*case-tac lista*)
apply *simp*
apply *simp*
done

lemma *normalize-call* [*simp*]:
normalize (*call init p return c*) = *call init p return* (λ*i t*. *normalize* (*c i t*))
by (*simp add*: *call-def*)

lemma *normalize-dynCall* [*simp*]:
normalize (*dynCall init p return c*) =
dynCall init p return (λ*s t*. *normalize* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *normalize-fcall* [*simp*]:
normalize (*fcall init p return result c*) =
fcall init p return result (λ*v*. *normalize* (*c v*))
by (*simp add*: *fcall-def*)

lemma *normalize-switch* [*simp*]:
normalize (*switch v Vcs*) = *switch v* (*map* (λ(*V*,*c*). (*V*,*normalize c*)) *Vcs*)
apply (*induct Vcs*)
apply *auto*
done

lemma *normalize-guaranteeStrip* [*simp*]:
normalize (*guaranteeStrip f g c*) = *guaranteeStrip f g* (*normalize c*)
by (*simp add*: *guaranteeStrip-def*)

lemma *normalize-guards* [*simp*]:
normalize (*guards gs c*) = *guards gs* (*normalize c*)
by (*induct gs*) *auto*

Sequencial composition with guards in the body is not preserved by normalize

lemma *normalize-while* [*simp*]:
normalize (*while gs b c*) = *guards gs*
(*While b* (*sequence Seq* (*flatten* (*normalize c*) @ *flatten* (*guards gs Skip*))))
by (*simp add*: *while-def*)

lemma *normalize-whileAnno* [*simp*]:
normalize (*whileAnno b I V c*) = *whileAnno b I V* (*normalize c*)
by (*simp add*: *whileAnno-def*)

lemma *normalize-whileAnnoG* [*simp*]:
normalize (*whileAnnoG gs b I V c*) = *guards gs*
(*While b* (*sequence Seq* (*flatten* (*normalize c*) @ *flatten* (*guards gs Skip*))))
by (*simp add*: *whileAnnoG-def*)

lemma *normalize-specAnno* [*simp*]:
normalize (*specAnno P c Q A*) = *specAnno P* (λ*s*. *normalize* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *normalize-simps* =
normalize.simps normalize-raise normalize-condCatch normalize-bind
normalize-block normalize-call normalize-dynCall normalize-fcall normalize-switch
normalize-guaranteeStrip normalize-guards
normalize-while normalize-whileAnno normalize-whileAnnoG normalize-specAnno

1.3.2 Stripping Guards: *strip-guards*

primrec *strip-guards*:: $'f$ *set* $\Rightarrow$ ($'s$,$'p$,$'f$) *com* $\Rightarrow$ ($'s$,$'p$,$'f$) *com*
where
strip-guards F Skip = *Skip* |
strip-guards F (*Basic f*) = *Basic f* |
strip-guards F (*Spec r*) = *Spec r* |

strip-guards F (*Seq* c_1 c_2) = (*Seq* (*strip-guards F* c_1) (*strip-guards F* c_2)) |
strip-guards F (*Cond b* c_1 c_2) = *Cond b* (*strip-guards F* c_1) (*strip-guards F* c_2) |
strip-guards F (*While b c*) = *While b* (*strip-guards F c*) |
strip-guards F (*Call p*) = *Call p* |
strip-guards F (*DynCom c*) = *DynCom* (λs. (*strip-guards F* (*c s*))) |
strip-guards F (*Guard f g c*) = (*if f* $\in$ *F then strip-guards F c*
else Guard f g (*strip-guards F c*)) |
strip-guards F Throw = *Throw* |
strip-guards F (*Catch* c_1 c_2) = *Catch* (*strip-guards F* c_1) (*strip-guards F* c_2)

definition *strip*:: *'f set* $\Rightarrow$
('*p* $\Rightarrow$ ('*s*,'*p*,'*f*) *com option*) $\Rightarrow$ ('*p* $\Rightarrow$ ('*s*,'*p*,'*f*) *com option*)
where *strip F* Γ = (λp. *map-option* (*strip-guards F*) (Γ *p*))

lemma *strip-simp* [*simp*]: (*strip F* Γ) *p* = *map-option* (*strip-guards F*) (Γ *p*)
by (*simp add*: *strip-def*)

lemma *dom-strip*: *dom* (*strip F* Γ) = *dom* Γ
by (*auto*)

lemma *strip-guards-idem*: *strip-guards F* (*strip-guards F c*) = *strip-guards F c*
by (*induct c*) *auto*

lemma *strip-idem*: *strip F* (*strip F* Γ) = *strip F* Γ
apply (*rule ext*)
apply (*case-tac* Γ *x*)
apply (*auto simp add*: *strip-guards-idem strip-def*)
done

lemma *strip-guards-raise* [*simp*]:
strip-guards F (*raise f*) = *raise f*
by (*simp add*: *raise-def*)

lemma *strip-guards-condCatch* [*simp*]:
strip-guards F (*condCatch c1 b c2*) =
condCatch (*strip-guards F c1*) *b* (*strip-guards F c2*)
by (*simp add*: *condCatch-def*)

lemma *strip-guards-bind* [*simp*]:
strip-guards F (*bind e c*) = *bind e* (λv. *strip-guards F* (*c v*))
by (*simp add*: *bind-def*)

lemma *strip-guards-bseq* [*simp*]:
strip-guards F (*bseq c1 c2*) = *bseq* (*strip-guards F c1*) (*strip-guards F c2*)
by (*simp add*: *bseq-def*)

lemma *strip-guards-block* [*simp*]:
strip-guards F (*block init bdy return c*) =

block init (*strip-guards F bdy*) *return* (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *block-def*)

lemma *strip-guards-call* [*simp*]:
strip-guards F (*call init p return c*) =
call init p return (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *call-def*)

lemma *strip-guards-dynCall* [*simp*]:
strip-guards F (*dynCall init p return c*) =
dynCall init p return (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *strip-guards-fcall* [*simp*]:
strip-guards F (*fcall init p return result c*) =
fcall init p return result (λv. *strip-guards F* (*c v*))
by (*simp add*: *fcall-def*)

lemma *strip-guards-switch* [*simp*]:
strip-guards F (*switch v Vc*) =
switch v (*map* (λ(*V*,*c*). (*V*,*strip-guards F c*)) *Vc*)
by (*induct Vc*) *auto*

lemma *strip-guards-guaranteeStrip* [*simp*]:
strip-guards F (*guaranteeStrip f g c*) =
(*if f* $\in$ *F then strip-guards F c*
else guaranteeStrip f g (*strip-guards F c*))
by (*simp add*: *guaranteeStrip-def*)

lemma *guaranteeStripPair-split-conv* [*simp*]: *case-prod c* (*guaranteeStripPair f g*)
= *c f g*
by (*simp add*: *guaranteeStripPair-def*)

lemma *strip-guards-guards* [*simp*]: *strip-guards F* (*guards gs c*) =
guards (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) (*strip-guards F c*)
by (*induct gs*) *auto*

lemma *strip-guards-while* [*simp*]:
strip-guards F (*while gs b c*) =
while (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) *b* (*strip-guards F c*)
by (*simp add*: *while-def*)

lemma *strip-guards-whileAnno* [*simp*]:
strip-guards F (*whileAnno b I V c*) = *whileAnno b I V* (*strip-guards F c*)
by (*simp add*: *whileAnno-def while-def*)

lemma *strip-guards-whileAnnoG* [*simp*]:
strip-guards F (*whileAnnoG gs b I V c*) =
whileAnnoG (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) *b I V* (*strip-guards F c*)

by (*simp add*: *whileAnnoG-def*)

lemma *strip-guards-specAnno* [*simp*]:
strip-guards F (*specAnno P c Q A*) =
specAnno P (λ*s*. *strip-guards F* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *strip-guards-simps* = *strip-guards.simps strip-guards-raise*
strip-guards-condCatch strip-guards-bind strip-guards-bseq strip-guards-block
strip-guards-dynCall strip-guards-fcall strip-guards-switch
strip-guards-guaranteeStrip guaranteeStripPair-split-conv strip-guards-guards
strip-guards-while strip-guards-whileAnno strip-guards-whileAnnoG
strip-guards-specAnno

1.3.3 Marking Guards: *mark-guards*

primrec *mark-guards*:: $'f \Rightarrow ('s,'p,'g)\ com \Rightarrow ('s,'p,'f)\ com$
where
mark-guards f Skip = *Skip* |
mark-guards f (*Basic g*) = *Basic g* |
mark-guards f (*Spec r*) = *Spec r* |
mark-guards f (*Seq* c_1 c_2) = (*Seq* (*mark-guards f* c_1) (*mark-guards f* c_2)) |
mark-guards f (*Cond b* c_1 c_2) = *Cond b* (*mark-guards f* c_1) (*mark-guards f* c_2) |
mark-guards f (*While b c*) = *While b* (*mark-guards f c*) |
mark-guards f (*Call p*) = *Call p* |
mark-guards f (*DynCom c*) = *DynCom* (λ*s*. (*mark-guards f* (*c s*))) |
mark-guards f (*Guard* f' *g c*) = *Guard f g* (*mark-guards f c*) |
mark-guards f Throw = *Throw* |
mark-guards f (*Catch* c_1 c_2) = *Catch* (*mark-guards f* c_1) (*mark-guards f* c_2)

lemma *mark-guards-raise*: *mark-guards f* (*raise g*) = *raise g*
by (*simp add*: *raise-def*)

lemma *mark-guards-condCatch* [*simp*]:
mark-guards f (*condCatch c1 b c2*) =
condCatch (*mark-guards f c1*) *b* (*mark-guards f c2*)
by (*simp add*: *condCatch-def*)

lemma *mark-guards-bind* [*simp*]:
mark-guards f (*bind e c*) = *bind e* (λ*v*. *mark-guards f* (*c v*))
by (*simp add*: *bind-def*)

lemma *mark-guards-bseq* [*simp*]:
mark-guards f (*bseq c1 c2*) = *bseq* (*mark-guards f c1*) (*mark-guards f c2*)
by (*simp add*: *bseq-def*)

lemma *mark-guards-block* [*simp*]:
mark-guards f (*block init bdy return c*) =
block init (*mark-guards f bdy*) *return* (λ*s t*. *mark-guards f* (*c s t*))

by (*simp add*: *block-def*)

lemma *mark-guards-call* [*simp*]:
mark-guards f (*call init p return c*) =
call init p return ($\lambda s\ t$. *mark-guards f* (*c s t*))
by (*simp add*: *call-def*)

lemma *mark-guards-dynCall* [*simp*]:
mark-guards f (*dynCall init p return c*) =
dynCall init p return ($\lambda s\ t$. *mark-guards f* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *mark-guards-fcall* [*simp*]:
mark-guards f (*fcall init p return result c*) =
fcall init p return result (λv. *mark-guards f* (*c v*))
by (*simp add*: *fcall-def*)

lemma *mark-guards-switch* [*simp*]:
mark-guards f (*switch v vs*) =
switch v (*map* ($\lambda(V,c)$. (V,*mark-guards f c*)) *vs*)
by (*induct vs*) *auto*

lemma *mark-guards-guaranteeStrip* [*simp*]:
mark-guards f (*guaranteeStrip f' g c*) = *guaranteeStrip f g* (*mark-guards f c*)
by (*simp add*: *guaranteeStrip-def*)

lemma *mark-guards-guards* [*simp*]:
mark-guards f (*guards gs c*) = *guards* (*map* ($\lambda(f',g)$. (f,g)) *gs*) (*mark-guards f c*)
by (*induct gs*) *auto*

lemma *mark-guards-while* [*simp*]:
mark-guards f (*while gs b c*) =
while (*map* ($\lambda(f',g)$. (f,g)) *gs*) *b* (*mark-guards f c*)
by (*simp add*: *while-def*)

lemma *mark-guards-whileAnno* [*simp*]:
mark-guards f (*whileAnno b I V c*) = *whileAnno b I V* (*mark-guards f c*)
by (*simp add*: *whileAnno-def while-def*)

lemma *mark-guards-whileAnnoG* [*simp*]:
mark-guards f (*whileAnnoG gs b I V c*) =
whileAnnoG (*map* ($\lambda(f',g)$. (f,g)) *gs*) *b I V* (*mark-guards f c*)
by (*simp add*: *whileAnno-def whileAnnoG-def while-def*)

lemma *mark-guards-specAnno* [*simp*]:
mark-guards f (*specAnno P c Q A*) =
specAnno P (λs. *mark-guards f* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *mark-guards-simps* = *mark-guards.simps mark-guards-raise mark-guards-condCatch mark-guards-bind mark-guards-bseq mark-guards-block mark-guards-dynCall mark-guards-fcall mark-guards-switch mark-guards-guaranteeStrip guaranteeStripPair-split-conv mark-guards-guards mark-guards-while mark-guards-whileAnno mark-guards-whileAnnoG mark-guards-specAnno*

definition *is-Guard*:: ('s,'p,'f) *com* ⇒ *bool*
where *is-Guard c* = (*case c of Guard f g c'* ⇒ *True* | - ⇒ *False*)
lemma *is-Guard-basic-simps* [*simp*]:
is-Guard Skip = *False*
is-Guard (*Basic f*) = *False*
is-Guard (*Spec r*) = *False*
is-Guard (*Seq c1 c2*) = *False*
is-Guard (*Cond b c1 c2*) = *False*
is-Guard (*While b c*) = *False*
is-Guard (*Call p*) = *False*
is-Guard (*DynCom C*) = *False*
is-Guard (*Guard F g c*) = *True*
is-Guard (*Throw*) = *False*
is-Guard (*Catch c1 c2*) = *False*
is-Guard (*raise f*) = *False*
is-Guard (*condCatch c1 b c2*) = *False*
is-Guard (*bind e cv*) = *False*
is-Guard (*bseq c1 c2*) = *False*
is-Guard (*block init bdy return cont*) = *False*
is-Guard (*call init p return cont*) = *False*
is-Guard (*dynCall init P return cont*) = *False*
is-Guard (*fcall init p return result cont'*) = *False*
is-Guard (*whileAnno b I V c*) = *False*
is-Guard (*guaranteeStrip F g c*) = *True*
by (*auto simp add*: *is-Guard-def raise-def condCatch-def bind-def bseq-def block-def call-def dynCall-def fcall-def whileAnno-def guaranteeStrip-def*)

lemma *is-Guard-switch* [*simp*]:
is-Guard (*switch v Vc*) = *False*
by (*induct Vc*) *auto*

lemmas *is-Guard-simps* = *is-Guard-basic-simps is-Guard-switch*

primrec *dest-Guard*:: ('s,'p,'f) *com* ⇒ ('f × 's *set* × ('s,'p,'f) *com*)
where *dest-Guard* (*Guard f g c*) = (*f*,*g*,*c*)

lemma *dest-Guard-guaranteeStrip* [*simp*]: *dest-Guard* (*guaranteeStrip f g c*) = (*f*,*g*,*c*)
by (*simp add*: *guaranteeStrip-def*)

lemmas *dest-Guard-simps* = *dest-Guard.simps* *dest-Guard-guaranteeStrip*

1.3.4 Merging Guards: *merge-guards*

primrec *merge-guards*:: $('s,'p,'f)$ *com* $\Rightarrow$ $('s,'p,'f)$ *com*
where
merge-guards Skip = *Skip* |
merge-guards (*Basic g*) = *Basic g* |
merge-guards (*Spec r*) = *Spec r* |
merge-guards (*Seq* c_1 c_2) = (*Seq* (*merge-guards* c_1) (*merge-guards* c_2)) |
merge-guards (*Cond b* c_1 c_2) = *Cond b* (*merge-guards* c_1) (*merge-guards* c_2) |
merge-guards (*While b c*) = *While b* (*merge-guards c*) |
merge-guards (*Call p*) = *Call p* |
merge-guards (*DynCom c*) = *DynCom* (λs. (*merge-guards* (*c s*))) |

merge-guards (*Guard f g c*) =
  (*let* c' = (*merge-guards c*)
   in if is-Guard c'
    then let (f',g',c'') = *dest-Guard* c'
      in if $f=f'$ *then Guard f* $(g \cap g')$ c''
           else Guard f g (*Guard* f' g' c'')
    else Guard f g c') |
merge-guards Throw = *Throw* |
merge-guards (*Catch* c_1 c_2) = *Catch* (*merge-guards* c_1) (*merge-guards* c_2)

lemma *merge-guards-res-Skip*: *merge-guards c* = *Skip* $\Longrightarrow$ *c* = *Skip*
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Basic*: *merge-guards c* = *Basic f* $\Longrightarrow$ *c* = *Basic f*
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Spec*: *merge-guards c* = *Spec r* $\Longrightarrow$ *c* = *Spec r*
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Seq*: *merge-guards c* = *Seq c1 c2* $\Longrightarrow$
  $\exists$ *c1' c2'*. *c* = *Seq c1' c2'* $\wedge$ *merge-guards c1'* = *c1* $\wedge$ *merge-guards c2'* = *c2*
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Cond*: *merge-guards c* = *Cond b c1 c2* $\Longrightarrow$
  $\exists$ *c1' c2'*. *c* = *Cond b c1' c2'* $\wedge$ *merge-guards c1'* = *c1* $\wedge$ *merge-guards c2'* = *c2*
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-While*: *merge-guards c* = *While b* c' $\Longrightarrow$
  $\exists$ c''. *c* = *While b* c'' $\wedge$ *merge-guards* c'' = c'
 by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Call*: *merge-guards c* = *Call p* $\Longrightarrow$ *c* = *Call p*

by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-DynCom*: *merge-guards c* = *DynCom c'* $\Longrightarrow$
$\exists\, c''.$ *c* = *DynCom c''* $\wedge$ ($\lambda s.$ (*merge-guards* (*c'' s*))) = *c'*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Throw*: *merge-guards c* = *Throw* $\Longrightarrow$ *c* = *Throw*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Catch*: *merge-guards c* = *Catch c1 c2* $\Longrightarrow$
$\exists$ *c1' c2'*. *c* = *Catch c1' c2'* $\wedge$ *merge-guards c1'* = *c1* $\wedge$ *merge-guards c2'* = *c2*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Guard*:
merge-guards c = *Guard f g c'* $\Longrightarrow$ $\exists$ *c'' f' g'*. *c* = *Guard f' g' c''*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemmas *merge-guards-res-simps* = *merge-guards-res-Skip merge-guards-res-Basic*
merge-guards-res-Spec merge-guards-res-Seq merge-guards-res-Cond
merge-guards-res-While merge-guards-res-Call
merge-guards-res-DynCom merge-guards-res-Throw merge-guards-res-Catch
merge-guards-res-Guard

lemma *merge-guards-raise*: *merge-guards* (*raise g*) = *raise g*
by (*simp add*: *raise-def*)

lemma *merge-guards-condCatch* [*simp*]:
merge-guards (*condCatch c1 b c2*) =
condCatch (*merge-guards c1*) *b* (*merge-guards c2*)
by (*simp add*: *condCatch-def*)

lemma *merge-guards-bind* [*simp*]:
merge-guards (*bind e c*) = *bind e* ($\lambda v.$ *merge-guards* (*c v*))
by (*simp add*: *bind-def*)

lemma *merge-guards-bseq* [*simp*]:
merge-guards (*bseq c1 c2*) = *bseq* (*merge-guards c1*) (*merge-guards c2*)
by (*simp add*: *bseq-def*)

lemma *merge-guards-block* [*simp*]:
merge-guards (*block init bdy return c*) =
block init (*merge-guards bdy*) *return* ($\lambda s\ t.$ *merge-guards* (*c s t*))
by (*simp add*: *block-def*)

lemma *merge-guards-call* [*simp*]:
merge-guards (*call init p return c*) =
call init p return ($\lambda s\ t.$ *merge-guards* (*c s t*))
by (*simp add*: *call-def*)

lemma *merge-guards-dynCall* [*simp*]:
  merge-guards (*dynCall init p return c*) =
    dynCall init p return (λ*s t*. *merge-guards* (*c s t*))
  by (*simp add*: *dynCall-def*)

lemma *merge-guards-fcall* [*simp*]:
  merge-guards (*fcall init p return result c*) =
    fcall init p return result (λ*v*. *merge-guards* (*c v*))
  by (*simp add*: *fcall-def*)

lemma *merge-guards-switch* [*simp*]:
  merge-guards (*switch v vs*) =
    switch v (*map* (λ(*V*,*c*). (*V*,*merge-guards c*)) *vs*)
  by (*induct vs*) *auto*

lemma *merge-guards-guaranteeStrip* [*simp*]:
  merge-guards (*guaranteeStrip f g c*) =
   (*let* c' = (*merge-guards c*)
    in if is-Guard c'
      then let (f',g',c') = *dest-Guard* c'
         in if f=f' *then Guard f* ($g \cap g'$) c'
                  else Guard f g (*Guard* f' g' c')
      else Guard f g c')
  by (*simp add*: *guaranteeStrip-def*)

lemma *merge-guards-whileAnno* [*simp*]:
 merge-guards (*whileAnno b I V c*) = *whileAnno b I V* (*merge-guards c*)
  by (*simp add*: *whileAnno-def while-def*)

lemma *merge-guards-specAnno* [*simp*]:
  merge-guards (*specAnno P c Q A*) =
   specAnno P (λ*s*. *merge-guards* (*c undefined*)) *Q A*
  by (*simp add*: *specAnno-def*)

merge-guards for guard-lists as in *guards*, *while* and *whileAnnoG* may have funny effects since the guard-list has to be merged with the body statement too.

lemmas *merge-guards-simps* = *merge-guards.simps merge-guards-raise*
 merge-guards-condCatch merge-guards-bind merge-guards-bseq merge-guards-block
 merge-guards-dynCall merge-guards-fcall merge-guards-switch
 merge-guards-guaranteeStrip merge-guards-whileAnno merge-guards-specAnno

primrec *noguards*:: ($'s$,$'p$,$'f$) *com* $\Rightarrow$ *bool*
where
noguards Skip = *True* |
noguards (*Basic f*) = *True* |
noguards (*Spec r*) = *True* |
noguards (*Seq* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2) |
noguards (*Cond b* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2) |

noguards (*While b c*) = (*noguards c*) |
noguards (*Call p*) = *True* |
noguards (*DynCom c*) = ($\forall$ *s. noguards* (*c s*)) |
noguards (*Guard f g c*) = *False* |
noguards Throw = *True* |
noguards (*Catch* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2)

lemma *noguards-strip-guards*: *noguards* (*strip-guards UNIV c*)
by (*induct c*) *auto*

primrec *nothrows*:: ('s,'p,'f) *com* $\Rightarrow$ *bool*
where
nothrows Skip = *True* |
nothrows (*Basic f*) = *True* |
nothrows (*Spec r*) = *True* |
nothrows (*Seq* c_1 c_2) = (*nothrows* c_1 $\wedge$ *nothrows* c_2) |
nothrows (*Cond b* c_1 c_2) = (*nothrows* c_1 $\wedge$ *nothrows* c_2) |
nothrows (*While b c*) = *nothrows c* |
nothrows (*Call p*) = *True* |
nothrows (*DynCom c*) = ($\forall$ *s. nothrows* (*c s*)) |
nothrows (*Guard f g c*) = *nothrows c* |
nothrows Throw = *False* |
nothrows (*Catch* c_1 c_2) = (*nothrows* c_1 $\wedge$ *nothrows* c_2)

1.3.5 Intersecting Guards: $c_1 \cap_g c_2$

inductive-set *com-rel* ::(('s,'p,'f) *com* $\times$ ('s,'p,'f) *com*) *set*
where
(*c1*, *Seq c1 c2*) $\in$ *com-rel*
| (*c2*, *Seq c1 c2*) $\in$ *com-rel*
| (*c1*, *Cond b c1 c2*) $\in$ *com-rel*
| (*c2*, *Cond b c1 c2*) $\in$ *com-rel*
| (*c*, *While b c*) $\in$ *com-rel*
| (*c x*, *DynCom c*) $\in$ *com-rel*
| (*c*, *Guard f g c*) $\in$ *com-rel*
| (*c1*, *Catch c1 c2*) $\in$ *com-rel*
| (*c2*, *Catch c1 c2*) $\in$ *com-rel*

inductive-cases *com-rel-elim-cases*:
(*c*, *Skip*) $\in$ *com-rel*
(*c*, *Basic f*) $\in$ *com-rel*
(*c*, *Spec r*) $\in$ *com-rel*
(*c*, *Seq c1 c2*) $\in$ *com-rel*
(*c*, *Cond b c1 c2*) $\in$ *com-rel*
(*c*, *While b c1*) $\in$ *com-rel*
(*c*, *Call p*) $\in$ *com-rel*
(*c*, *DynCom c1*) $\in$ *com-rel*
(*c*, *Guard f g c1*) $\in$ *com-rel*
(*c*, *Throw*) $\in$ *com-rel*

(c, *Catch c1 c2*) $\in$ *com-rel*

lemma *wf-com-rel*: *wf com-rel*
apply (*rule wfUNIVI*)
apply (*induct-tac x*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*,*simp*)
done

consts *inter-guards*:: ('s,'p,'f) *com* $\times$ ('s,'p,'f) *com* $\Rightarrow$ ('s,'p,'f) *com option*

abbreviation
inter-guards-syntax :: ('s,'p,'f) *com* $\Rightarrow$ ('s,'p,'f) *com* $\Rightarrow$ ('s,'p,'f) *com option*
(- $\cap_g$ - [*20*,*20*] *19*)
where $c \cap_g d$ == *inter-guards* (c,d)

recdef *inter-guards inv-image com-rel fst*
(*Skip* $\cap_g$ *Skip*) = *Some Skip*

(*Basic f1* $\cap_g$ *Basic f2*) = (*if* (*f1*=*f2*) *then Some* (*Basic f1*) *else None*)
(*Spec r1* $\cap_g$ *Spec r2*) = (*if* (*r1*=*r2*) *then Some* (*Spec r1*) *else None*)
(*Seq a1 a2* $\cap_g$ *Seq b1 b2*) =
  (*case* (*a1* $\cap_g$ *b1*) *of*
    None $\Rightarrow$ *None*
  | *Some c1* $\Rightarrow$ (*case* (*a2* $\cap_g$ *b2*) *of*
        None $\Rightarrow$ *None*
      | *Some c2* $\Rightarrow$ *Some* (*Seq c1 c2*)))

(*Cond cnd1 t1 e1* $\cap_g$ *Cond cnd2 t2 e2*) =
  (*if* (*cnd1*=*cnd2*)
   then (*case* (*t1* $\cap_g$ *t2*) *of*
      None $\Rightarrow$ *None*
    | *Some t* $\Rightarrow$ (*case* (*e1* $\cap_g$ *e2*) *of*
        None $\Rightarrow$ *None*
      | *Some e* $\Rightarrow$ *Some* (*Cond cnd1 t e*)))
   else None)

(*While cnd1 c1* $\cap_g$ *While cnd2 c2*) =

$(if\ (cnd1{=}cnd2\ )$
$\ \ then\ (case\ (c1\ \cap_g\ c2)\ of$
$\qquad None \Rightarrow None$
$\qquad |\ Some\ c \Rightarrow Some\ (While\ cnd1\ c))$
$\ \ else\ None)$

$(Call\ p1\ \cap_g\ Call\ p2) =$
$\ (if\ p1\ = p2$
$\ \ then\ Some\ (Call\ p1)$
$\ \ else\ None)$

$(DynCom\ P1\ \cap_g\ DynCom\ P2) =$
$\ (if\ (\forall s.\ ((P1\ s)\ \cap_g\ (P2\ s)) \neq None)$
$\ then\ Some\ (DynCom\ (\lambda s.\ \ the\ ((P1\ s)\ \cap_g\ (P2\ s))))$
$\ else\ None)$

$(Guard\ m1\ g1\ c1\ \cap_g\ Guard\ m2\ g2\ c2) =$
$\ (if\ m1{=}m2\ then$
$\quad (case\ (c1\ \cap_g\ c2)\ of$
$\qquad None \Rightarrow None$
$\quad\ |\ Some\ c \Rightarrow Some\ (Guard\ m1\ (g1\ \cap\ g2)\ c))$
$\ \ else\ None)$

$(Throw\ \cap_g\ Throw) = Some\ Throw$
$(Catch\ a1\ a2\ \cap_g\ Catch\ b1\ b2) =$
$\ (case\ (a1\ \cap_g\ b1)\ of$
$\quad None \Rightarrow None$
$\ \ |\ Some\ c1 \Rightarrow (case\ (a2\ \cap_g\ b2)\ of$
$\qquad\qquad None \Rightarrow None$
$\qquad\quad |\ Some\ c2 \Rightarrow Some\ (Catch\ c1\ c2)))$
$(c\ \cap_g\ d) = None$

(**hints** *cong add*: *option.case-cong if-cong*
recdef-wf: *wf-com-rel simp*: *com-rel.intros*)

lemma *inter-guards-strip-eq*:
$\bigwedge c.\ (c1\ \cap_g\ c2) = Some\ c\ \Longrightarrow$
$(strip\text{-}guards\ UNIV\ c = strip\text{-}guards\ UNIV\ c1) \wedge$
$(strip\text{-}guards\ UNIV\ c = strip\text{-}guards\ UNIV\ c2)$
apply (*induct c1 c2 rule*: *inter-guards.induct*)
prefer *8*
apply (*simp split*: *if-split-asm*)
apply *hypsubst*
apply *simp*
apply (*rule ext*)
apply (*erule-tac x=s* **in** *allE*, *erule exE*)
apply (*erule-tac x=s* **in** *allE*)
apply *fastforce*
apply (*fastforce split*: *option.splits if-split-asm*)+

done

lemma *inter-guards-sym*: $\bigwedge c.\ (c1 \cap_g c2) = Some\ c \Longrightarrow (c2 \cap_g c1) = Some\ c$
apply (*induct c1 c2 rule*: *inter-guards.induct*)
apply (*simp-all*)
prefer *7*
apply (*simp split*: *if-split-asm add*: *not-None-eq*)
apply (*rule conjI*)
apply (*clarsimp*)
apply (*rule ext*)
apply (*erule-tac x=s* **in** *allE*)+
apply *fastforce*
apply *fastforce*
apply (*fastforce split*: *option.splits if-split-asm*)+
done

lemma *inter-guards-Skip*: $(Skip \cap_g c2) = Some\ c = (c2{=}Skip \wedge c{=}Skip)$
  by (*cases c2*) *auto*

lemma *inter-guards-Basic*:
  $((Basic\ f) \cap_g c2) = Some\ c = (c2{=}Basic\ f \wedge c{=}Basic\ f)$
  by (*cases c2*) *auto*

lemma *inter-guards-Spec*:
  $((Spec\ r) \cap_g c2) = Some\ c = (c2{=}Spec\ r \wedge c{=}Spec\ r)$
  by (*cases c2*) *auto*

lemma *inter-guards-Seq*:
  $(Seq\ a1\ a2 \cap_g c2) = Some\ c =$
    $(\exists b1\ b2\ d1\ d2.\ c2{=}Seq\ b1\ b2 \wedge (a1 \cap_g b1) = Some\ d1 \wedge$
      $(a2 \cap_g b2) = Some\ d2 \wedge c{=}Seq\ d1\ d2)$
  by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-Cond*:
  $(Cond\ cnd\ t1\ e1 \cap_g c2) = Some\ c =$
    $(\exists t2\ e2\ t\ e.\ c2{=}Cond\ cnd\ t2\ e2 \wedge (t1 \cap_g t2) = Some\ t \wedge$
      $(e1 \cap_g e2) = Some\ e \wedge c{=}Cond\ cnd\ t\ e)$
  by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-While*:
 $(While\ cnd\ bdy1 \cap_g c2) = Some\ c =$
    $(\exists bdy2\ bdy.\ c2\ {=}While\ cnd\ bdy2 \wedge (bdy1 \cap_g bdy2) = Some\ bdy \wedge$
     $c{=}While\ cnd\ bdy)$
  by (*cases c2*) (*auto split*: *option.splits if-split-asm*)

lemma *inter-guards-Call*:
  $(Call\ p \cap_g c2) = Some\ c =$
    $(c2{=}Call\ p \wedge c{=}Call\ p)$

by (*cases c2*) (*auto split*: *if-split-asm*)

lemma *inter-guards-DynCom*:
  $(DynCom\ f1 \cap_g c2) = Some\ c =$
    $(\exists f2.\ c2{=}DynCom\ f2 \wedge (\forall s.\ ((f1\ s) \cap_g (f2\ s)) \neq None) \wedge$
     $c{=}DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_g (f2\ s))))$
  by (*cases c2*) (*auto split*: *if-split-asm*)

lemma *inter-guards-Guard*:
  $(Guard\ f\ g1\ bdy1 \cap_g c2) = Some\ c =$
    $(\exists g2\ bdy2\ bdy.\ c2{=}Guard\ f\ g2\ bdy2 \wedge (bdy1 \cap_g bdy2) = Some\ bdy \wedge$
      $c{=}Guard\ f\ (g1 \cap g2)\ bdy)$
  by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-Throw*:
  $(Throw \cap_g c2) = Some\ c = (c2{=}Throw \wedge c{=}Throw)$
  by (*cases c2*) *auto*

lemma *inter-guards-Catch*:
  $(Catch\ a1\ a2 \cap_g c2) = Some\ c =$
    $(\exists b1\ b2\ d1\ d2.\ c2{=}Catch\ b1\ b2 \wedge (a1 \cap_g b1) = Some\ d1 \wedge$
      $(a2 \cap_g b2) = Some\ d2 \wedge c{=}Catch\ d1\ d2)$
  by (*cases c2*) (*auto split*: *option.splits*)

lemmas *inter-guards-simps* = *inter-guards-Skip inter-guards-Basic inter-guards-Spec inter-guards-Seq inter-guards-Cond inter-guards-While inter-guards-Call inter-guards-DynCom inter-guards-Guard inter-guards-Throw inter-guards-Catch*

1.3.6 Subset on Guards: $c_1 \subseteq_g c_2$

consts *subseteq-guards*:: $('s,'p,'f)\ com \times ('s,'p,'f)\ com \Rightarrow bool$

abbreviation
  subseteq-guards-syntax :: $('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com \Rightarrow bool$
        $(\text{-} \subseteq_g \text{-}\ [20,20]\ 19)$
  where $c \subseteq_g d == subseteq\text{-}guards\ (c,d)$

recdef *subseteq-guards inv-image com-rel snd*
$(Skip \subseteq_g Skip) = True$
$(Basic\ f1 \subseteq_g Basic\ f2) = (f1{=}f2)$
$(Spec\ r1 \subseteq_g Spec\ r2) = (r1{=}r2)$
$(Seq\ a1\ a2 \subseteq_g Seq\ b1\ b2) = ((a1 \subseteq_g b1) \wedge (a2 \subseteq_g b2))$
$(Cond\ cnd1\ t1\ e1 \subseteq_g Cond\ cnd2\ t2\ e2) = ((cnd1{=}cnd2) \wedge (t1 \subseteq_g t2) \wedge (e1 \subseteq_g e2))$
$(While\ cnd1\ c1 \subseteq_g While\ cnd2\ c2) = ((cnd1{=}cnd2) \wedge (c1 \subseteq_g c2))$

$(Call\ p1 \subseteq_g Call\ p2) = (p1 = p2)$
$(DynCom\ P1 \subseteq_g DynCom\ P2) = (\forall s.\ ((P1\ s) \subseteq_g (P2\ s)))$
$(Guard\ m1\ g1\ c1 \subseteq_g Guard\ m2\ g2\ c2) =$
$((m1{=}m2 \wedge g1{=}g2 \wedge (c1 \subseteq_g c2)) \vee (Guard\ m1\ g1\ c1 \subseteq_g c2))$
$(c1 \subseteq_g Guard\ m2\ g2\ c2) = (c1 \subseteq_g c2)$

$(Throw \subseteq_g Throw) = True$
$(Catch\ a1\ a2 \subseteq_g Catch\ b1\ b2) = ((a1 \subseteq_g b1) \wedge (a2 \subseteq_g b2))$
$(c \subseteq_g d) = False$

(**hints** *cong add*: *if-cong*
recdef-wf: *wf-com-rel simp*: *com-rel.intros*)

lemma *subseteq-guards-Skip*:
$c \subseteq_g Skip \Longrightarrow c = Skip$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Basic*:
$c \subseteq_g Basic\ f \Longrightarrow c = Basic\ f$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Spec*:
$c \subseteq_g Spec\ r \Longrightarrow c = Spec\ r$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Seq*:
$c \subseteq_g Seq\ c1\ c2 \Longrightarrow \exists c1'\ c2'.\ c{=}Seq\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Cond*:
$c \subseteq_g Cond\ b\ c1\ c2 \Longrightarrow \exists c1'\ c2'.\ c{=}Cond\ b\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-While*:
$c \subseteq_g While\ b\ c' \Longrightarrow \exists c''.\ c{=}While\ b\ c'' \wedge (c'' \subseteq_g c')$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Call*:
$c \subseteq_g Call\ p \Longrightarrow c = Call\ p$
by (*cases c*) (*auto*)

lemma *subseteq-guards-DynCom*:
$c \subseteq_g DynCom\ C \Longrightarrow \exists C'.\ c{=}DynCom\ C' \wedge (\forall s.\ C'\ s \subseteq_g C\ s)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Guard*:
$c \subseteq_g Guard\ f\ g\ c' \Longrightarrow$
$(c \subseteq_g c') \vee (\exists c''.\ c{=}Guard\ f\ g\ c'' \wedge (c'' \subseteq_g c'))$

by (*cases c*) (*auto split*: *if-split-asm*)

lemma *subseteq-guards-Throw*:
$c \subseteq_g$ *Throw* $\Longrightarrow c =$ *Throw*
by (*cases c*) (*auto*)

lemma *subseteq-guards-Catch*:
$c \subseteq_g$ *Catch c1 c2* $\Longrightarrow \exists\, c1'\ c2'.\ c{=}Catch\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$
by (*cases c*) (*auto*)

lemmas *subseteq-guardsD* = *subseteq-guards-Skip subseteq-guards-Basic subseteq-guards-Spec subseteq-guards-Seq subseteq-guards-Cond subseteq-guards-While subseteq-guards-Call subseteq-guards-DynCom subseteq-guards-Guard subseteq-guards-Throw subseteq-guards-Catch*

lemma *subseteq-guards-Guard'*:
$Guard\ f\ b\ c \subseteq_g d \Longrightarrow \exists f'\ b'\ c'.\ d{=}Guard\ f'\ b'\ c'$
apply (*cases d*)
apply *auto*
done

lemma *subseteq-guards-refl*: $c \subseteq_g c$
by (*induct c*) *auto*

end

2 Big-Step Semantics for Simpl

theory *Semantic* **imports** *Language* **begin**

notation
restrict-map (-|- [*90*, *91*] *90*)

datatype ('*s*,'*f*) *xstate* = *Normal* '*s* | *Abrupt* '*s* | *Fault* '*f* | *Stuck*

definition *isAbr*::('*s*,'*f*) *xstate* ⇒ *bool*
where *isAbr S* = (∃ *s*. *S*=*Abrupt s*)

lemma *isAbr-simps* [*simp*]:
isAbr (*Normal s*) = *False*
isAbr (*Abrupt s*) = *True*
isAbr (*Fault f*) = *False*
isAbr Stuck = *False*
by (*auto simp add*: *isAbr-def*)

lemma *isAbrE* [*consumes 1*, *elim?*]: $[\![isAbr\ S;\ \bigwedge s.\ S{=}Abrupt\ s \Longrightarrow P]\!] \Longrightarrow P$

by (*auto simp add*: *isAbr-def*)

lemma *not-isAbrD*:
$\neg$ *isAbr s* $\Longrightarrow$ ($\exists s'$. *s*=*Normal s'*) $\vee$ *s* = *Stuck* $\vee$ ($\exists f$. *s*=*Fault f*)
by (*cases s*) *auto*

definition *isFault*:: ('*s*,'*f*) *xstate* $\Rightarrow$ *bool*
where *isFault S* = ($\exists f$. *S*=*Fault f*)

lemma *isFault-simps* [*simp*]:
isFault (*Normal s*) = *False*
isFault (*Abrupt s*) = *False*
isFault (*Fault f*) = *True*
isFault Stuck = *False*
by (*auto simp add*: *isFault-def*)

lemma *isFaultE* [*consumes 1*, *elim?*]: $[\![$*isFault s*; $\bigwedge f$. *s*=*Fault f* $\Longrightarrow$ *P*$]\!]$ $\Longrightarrow$ *P*
by (*auto simp add*: *isFault-def*)

lemma *not-isFault-iff*: ($\neg$ *isFault t*) = ($\forall f$. *t* $\neq$ *Fault f*)
by (*auto elim*: *isFaultE*)

2.1 Big-Step Execution: $\Gamma\vdash\langle c,\ s\rangle \Rightarrow t$

The procedure environment

type-synonym ('*s*,'*p*,'*f*) *body* = '*p* $\Rightarrow$ ('*s*,'*p*,'*f*) *com option*

inductive
exec::[('*s*,'*p*,'*f*) *body*,('*s*,'*p*,'*f*) *com*,('*s*,'*f*) *xstate*,('*s*,'*f*) *xstate*]
$\Rightarrow$ *bool* (-$\vdash$ $\langle$-,-$\rangle$ $\Rightarrow$ - [*60*,*20*,*98*,*98*] *89*)
for Γ::('*s*,'*p*,'*f*) *body*
where
Skip: $\Gamma\vdash\langle$*Skip*,*Normal s*$\rangle$ $\Rightarrow$ *Normal s*

| *Guard*: $[\![$*s*$\in$*g*; $\Gamma\vdash\langle$*c*,*Normal s*$\rangle$ $\Rightarrow$ *t*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*Guard f g c*,*Normal s*$\rangle$ $\Rightarrow$ *t*

| *GuardFault*: *s*$\notin$*g* $\Longrightarrow$ $\Gamma\vdash\langle$*Guard f g c*,*Normal s*$\rangle$ $\Rightarrow$ *Fault f*

| *FaultProp* [*intro*,*simp*]: $\Gamma\vdash\langle$*c*,*Fault f*$\rangle$ $\Rightarrow$ *Fault f*

| *Basic*: $\Gamma\vdash\langle$*Basic f*,*Normal s*$\rangle$ $\Rightarrow$ *Normal* (*f s*)

| *Spec*: (*s*,*t*) $\in$ *r*
$\Longrightarrow$
$\Gamma\vdash\langle$*Spec r*,*Normal s*$\rangle$ $\Rightarrow$ *Normal t*

| *SpecStuck*: $\forall t$. (*s*,*t*) $\notin$ *r*

$\Longrightarrow$
$\Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\ Stuck$

| *Seq*: $[\![\Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\ s'; \Gamma\vdash\langle c_2,s'\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Seq\ c_1\ c_2,Normal\ s\rangle \Rightarrow\ t$

| *CondTrue*: $[\![s \in b; \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Cond\ b\ c_1\ c_2,Normal\ s\rangle \Rightarrow\ t$

| *CondFalse*: $[\![s \notin b; \Gamma\vdash\langle c_2,Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Cond\ b\ c_1\ c_2,Normal\ s\rangle \Rightarrow\ t$

| *WhileTrue*: $[\![s \in b; \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\ s'; \Gamma\vdash\langle While\ b\ c,s'\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\ t$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\ Normal\ s$

| *Call*: $[\![\Gamma\ p=Some\ bdy;\Gamma\vdash\langle bdy,Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\ t$

| *CallUndefined*: $[\![\Gamma\ p=None]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\ Stuck$

| *StuckProp* [*intro*,*simp*]: $\Gamma\vdash\langle c,Stuck\rangle \Rightarrow\ Stuck$

| *DynCom*: $[\![\Gamma\vdash\langle(c\ s),Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\ t$

| *Throw*: $\Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\ Abrupt\ s$

| *AbruptProp* [*intro*,*simp*]: $\Gamma\vdash\langle c,Abrupt\ s\rangle \Rightarrow\ Abrupt\ s$

| *CatchMatch*: $[\![\Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\ Abrupt\ s'; \Gamma\vdash\langle c_2,Normal\ s'\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\ t$
| *CatchMiss*: $[\![\Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\ t; \neg isAbr\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\ t$

inductive-cases *exec-elim-cases* [*cases set*]:

$\Gamma\vdash\langle c,Fault\ f\rangle \Rightarrow\ t$
$\Gamma\vdash\langle c,Stuck\rangle \Rightarrow\ t$
$\Gamma\vdash\langle c,Abrupt\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Skip,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Guard\ f\ g\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Basic\ f,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Spec\ r,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Cond\ b\ c1\ c2,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle While\ b\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Call\ p,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle DynCom\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Throw,s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Catch\ c1\ c2,s\rangle \Rightarrow\ t$

inductive-cases *exec-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash\langle c,Fault\ f\rangle \Rightarrow\ t$
$\Gamma\vdash\langle c,Stuck\rangle \Rightarrow\ t$
$\Gamma\vdash\langle c,Abrupt\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Skip,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Basic\ f,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$

lemma *exec-block*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t;\ \Gamma\vdash\langle c\ s\ t,Normal\ (return\ s\ t)\rangle \Rightarrow\ u]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle \Rightarrow\ u$
apply (*unfold block-def*)
by (*fastforce intro*: *exec.intros*)

lemma *exec-blockAbrupt*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle \Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*unfold block-def*)
by (*fastforce intro*: *exec.intros*)

lemma *exec-blockFault*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$

$\Gamma\vdash\langle$*block init bdy return c*,*Normal s*$\rangle \Rightarrow$ *Fault f*
apply (*unfold block-def*)
by (*fastforce intro*: *exec.intros*)

lemma *exec-blockStuck*:
$[\![\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Stuck*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*block init bdy return c*,*Normal s*$\rangle \Rightarrow$ *Stuck*
apply (*unfold block-def*)
by (*fastforce intro*: *exec.intros*)

lemma *exec-call*:
$[\![\Gamma$ *p*=*Some bdy*;$\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Normal t*; $\Gamma\vdash\langle$*c s t*,*Normal* (*return s t*)$\rangle \Rightarrow$ *u*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle \Rightarrow$ *u*
apply (*simp add*: *call-def*)
apply (*rule exec-block*)
apply (*erule* (*1*) *Call*)
apply *assumption*
done

lemma *exec-callAbrupt*:
$[\![\Gamma$ *p*=*Some bdy*;$\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Abrupt t*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle \Rightarrow$ *Abrupt* (*return s t*)
apply (*simp add*: *call-def*)
apply (*rule exec-blockAbrupt*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callFault*:
$[\![\Gamma$ *p*=*Some bdy*; $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Fault f*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle \Rightarrow$ *Fault f*
apply (*simp add*: *call-def*)
apply (*rule exec-blockFault*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callStuck*:
$[\![\Gamma$ *p*=*Some bdy*; $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Stuck*$]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle \Rightarrow$ *Stuck*
apply (*simp add*: *call-def*)
apply (*rule exec-blockStuck*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callUndefined*:
  $[\![\Gamma\ p{=}None]\!]$
  $\Longrightarrow$
  $\Gamma\vdash\langle call\ init\ p\ return\ c{,}Normal\ s\rangle \Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule exec-blockStuck*)
apply (*erule CallUndefined*)
done

lemma *Fault-end*: **assumes** *exec*: $\Gamma\vdash\langle c{,}s\rangle \Rightarrow\ t$ **and** *s*: $s{=}Fault\ f$
  shows $t{=}Fault\ f$
using *exec s* **by** (*induct*) *auto*

lemma *Stuck-end*: **assumes** *exec*: $\Gamma\vdash\langle c{,}s\rangle \Rightarrow\ t$ **and** *s*: $s{=}Stuck$
  shows $t{=}Stuck$
using *exec s* **by** (*induct*) *auto*

lemma *Abrupt-end*: **assumes** *exec*: $\Gamma\vdash\langle c{,}s\rangle \Rightarrow\ t$ **and** *s*: $s{=}Abrupt\ s'$
  shows $t{=}Abrupt\ s'$
using *exec s* **by** (*induct*) *auto*

lemma *exec-Call-body-aux*:
  $\Gamma\ p{=}Some\ bdy \Longrightarrow$
  $\Gamma\vdash\langle Call\ p{,}s\rangle \Rightarrow t = \Gamma\vdash\langle bdy{,}s\rangle \Rightarrow t$
apply (*rule*)
apply (*fastforce elim*: *exec-elim-cases*)
apply (*cases s*)
apply (*cases t*)
apply (*auto intro*: *exec.intros dest*: *Fault-end Stuck-end Abrupt-end*)
done

lemma *exec-Call-body'*:
  $p \in dom\ \Gamma \Longrightarrow$
  $\Gamma\vdash\langle Call\ p{,}s\rangle \Rightarrow t = \Gamma\vdash\langle the\ (\Gamma\ p){,}s\rangle \Rightarrow t$
  apply *clarsimp*
  by (*rule exec-Call-body-aux*)

lemma *exec-block-Normal-elim* [*consumes 1*]:
assumes *exec-block*: $\Gamma\vdash\langle block\ init\ bdy\ return\ c{,}Normal\ s\rangle \Rightarrow\ t$
assumes *Normal*:
$\bigwedge t'.$
  $[\![\Gamma\vdash\langle bdy{,}Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t';$
  $\Gamma\vdash\langle c\ s\ t'{,}Normal\ (return\ s\ t')\rangle \Rightarrow\ t]\!]$
  $\Longrightarrow P$
assumes *Abrupt*:

$\bigwedge t'$.
  $[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t';$
   $t = Abrupt\ (return\ s\ t')]\!]$
  $\Longrightarrow P$
assumes *Fault*:
$\bigwedge f$.
  $[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f;$
   $t = Fault\ f]\!]$
  $\Longrightarrow P$
assumes *Stuck*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Stuck;$
   $t = Stuck]\!]$
  $\Longrightarrow P$
assumes
$[\![\Gamma\ p = None;\ t = Stuck]\!] \Longrightarrow P$
shows P
 using *exec-block*
apply (*unfold block-def*)
apply (*elim exec-Normal-elim-cases*)
apply *simp-all*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*drule Abrupt-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt,assumption+*)
apply (*drule Fault-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)
apply *simp*
apply (*drule Stuck-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)
apply *simp*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Normal, assumption+*)
apply (*drule Fault-end*) **apply** *simp*
apply (*rule Fault,assumption+*)
apply (*drule Stuck-end*) **apply** *simp*
apply (*rule Stuck,assumption+*)
done

lemma *exec-call-Normal-elim* [*consumes 1*]:
assumes *exec-call*: $\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle \Rightarrow\ t$
assumes *Normal*:
$\bigwedge bdy\ t'$.

$\llbracket\Gamma\ p = \mathit{Some}\ \mathit{bdy};\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow\ \mathit{Normal}\ t';$
$\Gamma\vdash\langle c\ s\ t',\mathit{Normal}\ (\mathit{return}\ s\ t')\rangle \Rightarrow\ t\rrbracket$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge \mathit{bdy}\ t'.$
$\llbracket\Gamma\ p = \mathit{Some}\ \mathit{bdy};\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow\ \mathit{Abrupt}\ t';$
$t = \mathit{Abrupt}\ (\mathit{return}\ s\ t')\rrbracket$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge \mathit{bdy}\ f.$
$\llbracket\Gamma\ p = \mathit{Some}\ \mathit{bdy};\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow\ \mathit{Fault}\ f;$
$t = \mathit{Fault}\ f\rrbracket$
$\Longrightarrow P$
assumes *Stuck*:
$\bigwedge \mathit{bdy}.$
$\llbracket\Gamma\ p = \mathit{Some}\ \mathit{bdy};\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow\ \mathit{Stuck};$
$t = \mathit{Stuck}\rrbracket$
$\Longrightarrow P$
assumes *Undef*:
$\llbracket\Gamma\ p = \mathit{None};\ t = \mathit{Stuck}\rrbracket \Longrightarrow P$
shows P
using *exec-call*
apply (*unfold call-def*)
apply (*cases* Γ *p*)
apply (*erule exec-block-Normal-elim*)
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Undef*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
apply (*erule exec-block-Normal-elim*)
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*,*assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt*,*assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*

apply (*rule Fault*, *assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Stuck*,*assumption*,*assumption*,*assumption*)
apply *simp*
apply (*rule Undef*,*assumption*+)
done

lemma *exec-dynCall*:
$[\![\Gamma\vdash\langle call\ init\ (p\ s)\ return\ c,Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle dynCall\ init\ p\ return\ c,Normal\ s\rangle \Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

lemma *exec-dynCall-Normal-elim*:
assumes *exec*: $\Gamma\vdash\langle dynCall\ init\ p\ return\ c,Normal\ s\rangle \Rightarrow\ t$
assumes *call*: $\Gamma\vdash\langle call\ init\ (p\ s)\ return\ c,Normal\ s\rangle \Rightarrow\ t \Longrightarrow P$
shows *P*
using *exec*
apply (*simp add*: *dynCall-def*)
apply (*erule exec-Normal-elim-cases*)
apply (*rule call*,*assumption*)
done

lemma *exec-Call-body*:
$\Gamma\ p=Some\ bdy \Longrightarrow$
$\Gamma\vdash\langle Call\ p,s\rangle \Rightarrow\ t = \Gamma\vdash\langle the\ (\Gamma\ p),s\rangle \Rightarrow\ t$
apply (*rule*)
apply (*fastforce elim*: *exec-elim-cases*)
apply (*cases s*)
apply (*cases t*)
apply (*fastforce intro*: *exec.intros dest*: *Fault-end Abrupt-end Stuck-end*)+
done

lemma *exec-Seq'*: $[\![\Gamma\vdash\langle c1,s\rangle \Rightarrow\ s';\ \Gamma\vdash\langle c2,s'\rangle \Rightarrow\ s'']\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\ s''$
apply (*cases s*)
apply (*fastforce intro*: *exec.intros*)
apply (*fastforce dest*: *Abrupt-end*)
apply (*fastforce dest*: *Fault-end*)
apply (*fastforce dest*: *Stuck-end*)
done

lemma *exec-assoc*: $\Gamma\vdash\langle Seq\ c1\ (Seq\ c2\ c3),s\rangle \Rightarrow\ t = \Gamma\vdash\langle Seq\ (Seq\ c1\ c2)\ c3,s\rangle \Rightarrow\ t$
by (*blast elim*!: *exec-elim-cases intro*: *exec-Seq′*)

2.2 Big-Step Execution with Recursion Limit: $\Gamma\vdash\langle c,\ s\rangle =n\Rightarrow t$

inductive *execn*::$[('s,'p,'f)\ body,('s,'p,'f)\ com,('s,'f)\ xstate,nat,('s,'f)\ xstate]$
$\Rightarrow bool$ ($\text{-}\vdash \langle\text{-},\text{-}\rangle$ $=\text{-}\Rightarrow$ - $[60,20,98,65,98]$ 89)
for Γ::$('s,'p,'f)\ body$
where
Skip: $\Gamma\vdash\langle Skip,Normal\ s\rangle =n\Rightarrow\ Normal\ s$
| *Guard*: $[\![s\in g;\ \Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ t$

| *GuardFault*: $s\notin g \Longrightarrow \Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ Fault\ f$

| *FaultProp* [*intro*,*simp*]: $\Gamma\vdash\langle c,Fault\ f\rangle =n\Rightarrow\ Fault\ f$

| *Basic*: $\Gamma\vdash\langle Basic\ f,Normal\ s\rangle =n\Rightarrow\ Normal\ (f\ s)$

| *Spec*: $(s,t) \in r$
$\Longrightarrow$
$\Gamma\vdash\langle Spec\ r,Normal\ s\rangle =n\Rightarrow\ Normal\ t$

| *SpecStuck*: $\forall\, t.\ (s,t) \notin r$
$\Longrightarrow$
$\Gamma\vdash\langle Spec\ r,Normal\ s\rangle =n\Rightarrow\ Stuck$

| *Seq*: $[\![\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow\ s';\ \Gamma\vdash\langle c_2,s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Seq\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$

| *CondTrue*: $[\![s \in b;\ \Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Cond\ b\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$

| *CondFalse*: $[\![s \notin b;\ \Gamma\vdash\langle c_2,Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Cond\ b\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$

| *WhileTrue*: $[\![s \in b;\ \Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ s';$
$\Gamma\vdash\langle While\ b\ c,s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle =n\Rightarrow\ t$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$

$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ \mathit{Normal}\ s$

| *Call*: $[\![\Gamma\ p=\mathit{Some}\ \mathit{bdy};\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{Call}\ p\ ,\mathit{Normal}\ s\rangle\ =\mathit{Suc}\ n\Rightarrow\ \ t$

| *CallUndefined*: $[\![\Gamma\ p=\mathit{None}]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{Call}\ p\ ,\mathit{Normal}\ s\rangle\ =\mathit{Suc}\ n\Rightarrow\ \ \mathit{Stuck}$

| *StuckProp* [*intro*,*simp*]: $\Gamma\vdash\langle c,\mathit{Stuck}\rangle\ =n\Rightarrow\ \ \mathit{Stuck}$

| *DynCom*: $[\![\Gamma\vdash\langle (c\ s),\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{DynCom}\ c,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t$

| *Throw*: $\Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ \mathit{Abrupt}\ s$

| *AbruptProp* [*intro*,*simp*]: $\Gamma\vdash\langle c,\mathit{Abrupt}\ s\rangle\ =n\Rightarrow\ \ \mathit{Abrupt}\ s$

| *CatchMatch*: $[\![\Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ \mathit{Abrupt}\ s';\ \Gamma\vdash\langle c_2,\mathit{Normal}\ s'\rangle\ =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ \ c_2,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ t$
| *CatchMiss*: $[\![\Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t;\ \neg \mathit{isAbr}\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ \ c_2,\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t$

inductive-cases *execn-elim-cases* [*cases set*]:
$\Gamma\vdash\langle c,\mathit{Fault}\ f\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle c,\mathit{Stuck}\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle c,\mathit{Abrupt}\ s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Skip},s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Seq}\ c1\ \ c2,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Guard}\ f\ g\ \ c,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Basic}\ f,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Spec}\ r,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Cond}\ b\ \ c1\ \ c2,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{While}\ b\ \ c,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Call}\ p\ \ ,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{DynCom}\ c,s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Throw},s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Catch}\ c1\ \ c2,s\rangle\ =n\Rightarrow\ \ t$

inductive-cases *execn-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash\langle c,\mathit{Fault}\ f\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle c,\mathit{Stuck}\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle c,\mathit{Abrupt}\ s\rangle\ =n\Rightarrow\ \ t$
$\Gamma\vdash\langle \mathit{Skip},\mathit{Normal}\ s\rangle\ =n\Rightarrow\ \ t$

$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Basic\ f,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Spec\ r,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Throw,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$

lemma *execn-Skip'*: $\Gamma\vdash\langle Skip,t\rangle =n\Rightarrow t$
by (*cases t*) (*auto intro*: *execn.intros*)

lemma *execn-Fault-end*: **assumes** *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: $s=Fault\ f$
shows $t=Fault\ f$
using *exec s* **by** (*induct*) *auto*

lemma *execn-Stuck-end*: **assumes** *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: $s=Stuck$
shows $t=Stuck$
using *exec s* **by** (*induct*) *auto*

lemma *execn-Abrupt-end*: **assumes** *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: $s=Abrupt\ s'$
shows $t=Abrupt\ s'$
using *exec s* **by** (*induct*) *auto*

lemma *execn-block*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t;\ \Gamma\vdash\langle c\ s\ t,Normal\ (return\ s\ t)\rangle =n\Rightarrow$
$u]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle =n\Rightarrow\ u$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockAbrupt*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle =n\Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockFault*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle =n\Rightarrow\ Fault\ f$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockStuck*:

$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle =n\Rightarrow\ Stuck$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-call*:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t;$
$\Gamma\vdash\langle c\ s\ t,Normal\ (return\ s\ t)\rangle =Suc\ n\Rightarrow\ u]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ u$
apply (*simp add*: *call-def*)
apply (*rule execn-block*)
apply (*erule* (*1*) *Call*)
apply *assumption*
done

lemma *execn-callAbrupt*:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*simp add*: *call-def*)
apply (*rule execn-blockAbrupt*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callFault*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Fault\ f$
apply (*simp add*: *call-def*)
apply (*rule execn-blockFault*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callStuck*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule execn-blockStuck*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callUndefined*:
$[\![\Gamma\ p=None]\!]$
$\Longrightarrow$

$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule execn-blockStuck*)
apply (*erule CallUndefined*)
done

lemma *execn-block-Normal-elim* [*consumes 1*]:
assumes *execn-block*: $\Gamma\vdash\langle block\ init\ bdy\ return\ c,Normal\ s\rangle =n\Rightarrow\ t$
assumes *Normal*:
$\bigwedge t'.$
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t';$
$\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge t'.$
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t';$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge f.$
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$[\![\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes *Undef*:
$[\![\Gamma\ p = None;\ t = Stuck]\!] \Longrightarrow P$
shows *P*
using *execn-block*
apply (*unfold block-def*)
apply (*elim execn-Normal-elim-cases*)
apply *simp-all*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*drule execn-Abrupt-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt*,*assumption+*)
apply (*drule execn-Fault-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*drule execn-Stuck-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*case-tac s'*)

apply *simp-all*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*,*assumption+*)
apply (*drule execn-Fault-end*) **apply** *simp*
apply (*rule Fault*,*assumption+*)
apply (*drule execn-Stuck-end*) **apply** *simp*
apply (*rule Stuck*,*assumption+*)
done

lemma *execn-call-Normal-elim* [*consumes 1*]:
assumes *exec-call*: $\Gamma\vdash\langle call\ init\ p\ return\ c, Normal\ s\rangle =n\Rightarrow\ t$
assumes *Normal*:
$\bigwedge bdy\ i\ t'.$
$[\![\Gamma\ p = Some\ bdy; \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle =i\Rightarrow\ Normal\ t';$
$\Gamma\vdash\langle c\ s\ t', Normal\ (return\ s\ t')\rangle =Suc\ i\Rightarrow\ t;\ n = Suc\ i]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge bdy\ i\ t'.$
$[\![\Gamma\ p = Some\ bdy; \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle =i\Rightarrow\ Abrupt\ t';\ n = Suc\ i;$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge bdy\ i\ f.$
$[\![\Gamma\ p = Some\ bdy; \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle =i\Rightarrow\ Fault\ f;\ n = Suc\ i;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$\bigwedge bdy\ i.$
$[\![\Gamma\ p = Some\ bdy; \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle =i\Rightarrow\ Stuck;\ n = Suc\ i;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes *Undef*:
$\bigwedge i.\ [\![\Gamma\ p = None;\ n = Suc\ i;\ t = Stuck]\!] \Longrightarrow P$
shows *P*
using *exec-call*
apply (*unfold call-def*)
apply (*cases n*)
apply (*simp only*: *block-def*)
apply (*fastforce elim*: *execn-Normal-elim-cases*)
apply (*cases* Γ *p*)
apply (*erule execn-block-Normal-elim*)
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)

apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Undef*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
apply (*erule execn-block-Normal-elim*)
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Fault*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Stuck*,*assumption*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
done

lemma *execn-dynCall*:
$[\![\Gamma\vdash\langle call\ init\ (p\ s)\ return\ c,Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle dynCall\ init\ p\ return\ c,Normal\ s\rangle =n\Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

lemma *execn-dynCall-Normal-elim*:
assumes *exec*: $\Gamma\vdash\langle dynCall\ init\ p\ return\ c,Normal\ s\rangle =n\Rightarrow\ t$
assumes $\Gamma\vdash\langle call\ init\ (p\ s)\ return\ c,Normal\ s\rangle =n\Rightarrow\ t \Longrightarrow P$
shows *P*
using *exec*
apply (*simp add*: *dynCall-def*)
apply (*erule execn-Normal-elim-cases*)
apply *fact*
done

lemma *execn-Seq'*:
$[\![\Gamma\vdash\langle c1,s\rangle =n\Rightarrow\ s';\ \Gamma\vdash\langle c2,s'\rangle =n\Rightarrow\ s'']\!]$

$\Longrightarrow$
$\Gamma\vdash\langle Seq\ c1\ c2,s\rangle =n\Rightarrow\ s''$
apply (*cases s*)
apply (*fastforce intro*: *execn.intros*)
apply (*fastforce dest*: *execn-Abrupt-end*)
apply (*fastforce dest*: *execn-Fault-end*)
apply (*fastforce dest*: *execn-Stuck-end*)
done

lemma *execn-mono*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$
shows $\bigwedge m.\ n \leq m \Longrightarrow \Gamma\vdash\langle c,s\rangle =m\Rightarrow\ t$
using *exec*
by (*induct*) (*auto intro*: *execn.intros dest*: *Suc-le-D*)

lemma *execn-Suc*:
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t \Longrightarrow \Gamma\vdash\langle c,s\rangle =Suc\ n\Rightarrow\ t$
by (*rule execn-mono* [*OF* - *le-refl* [*THEN le-SucI*]])

lemma *execn-assoc*:
$\Gamma\vdash\langle Seq\ c1\ (Seq\ c2\ c3),s\rangle =n\Rightarrow\ t = \Gamma\vdash\langle Seq\ (Seq\ c1\ c2)\ c3,s\rangle =n\Rightarrow\ t$
by (*auto elim*!: *execn-elim-cases intro*: *execn-Seq'*)

lemma *execn-to-exec*:
assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *execn*
by *induct* (*auto intro*: *exec.intros*)

lemma *exec-to-execn*:
assumes *execn*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
shows $\exists n.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$
using *execn*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Guard* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Spec* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *SpecStuck* **thus** *?case* **by** (*iprover intro*: *execn.intros*)

next
  case (*Seq c1 s s' c2 s''*)
  then obtain *n m* **where**
    $\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash\langle c2,s'\rangle =m\Rightarrow\ s''$
    by *blast*
  then have
    $\Gamma\vdash\langle c1,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$
    $\Gamma\vdash\langle c2,s'\rangle =max\ n\ m\Rightarrow\ s''$
    by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
  thus *?case*
    by (*iprover intro*: *execn.intros*)
next
  case *CondTrue* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *CondFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case (*WhileTrue s b c s' s''*)
  then obtain *n m* **where**
    $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash\langle While\ b\ c,s'\rangle =m\Rightarrow\ s''$
    by *blast*
  then have
    $\Gamma\vdash\langle c,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$ $\Gamma\vdash\langle While\ b\ c,s'\rangle =max\ n\ m\Rightarrow\ s''$
    by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
  with *WhileTrue*
  show *?case*
    by (*iprover intro*: *execn.intros*)
next
  case *WhileFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *Call* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *CallUndefined* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *StuckProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *DynCom* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *Throw* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case *AbruptProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
  case (*CatchMatch c1 s s' c2 s''*)
  then obtain *n m* **where**
    $\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow\ Abrupt\ s'$ $\Gamma\vdash\langle c2,Normal\ s'\rangle =m\Rightarrow\ s''$
    by *blast*
  then have
    $\Gamma\vdash\langle c1,Normal\ s\rangle =max\ n\ m\Rightarrow\ Abrupt\ s'$
    $\Gamma\vdash\langle c2,Normal\ s'\rangle =max\ n\ m\Rightarrow\ s''$
    by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)

with *CatchMatch.hyps* **show** *?case*
by (*iprover intro*: *execn.intros*)
next
case *CatchMiss* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
qed

theorem *exec-iff-execn*: $(\Gamma\vdash\langle c,s\rangle \Rightarrow t) = (\exists\, n.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t)$
by (*iprover intro*: *exec-to-execn execn-to-exec*)

definition *nfinal-notin*:: $('s,'p,'f)$ *body* $\Rightarrow$ $('s,'p,'f)$ *com* $\Rightarrow$ $('s,'f)$ *xstate* $\Rightarrow$ *nat*
$\Rightarrow$ $('s,'f)$ *xstate set* $\Rightarrow$ *bool*
(-$\vdash$ $\langle$-,-$\rangle$ =-$\Rightarrow\notin$- [*60*,*20*,*98*,*65*,*60*] *89*) **where**
$\Gamma\vdash\ \langle c,s\rangle =n\Rightarrow\notin T = (\forall\, t.\ \Gamma\vdash\ \langle c,s\rangle =n\Rightarrow\ t \longrightarrow t\notin T)$

definition *final-notin*:: $('s,'p,'f)$ *body* $\Rightarrow$ $('s,'p,'f)$ *com* $\Rightarrow$ $('s,'f)$ *xstate*
$\Rightarrow$ $('s,'f)$ *xstate set* $\Rightarrow$ *bool*
(-$\vdash$ $\langle$-,-$\rangle$ $\Rightarrow\notin$- [*60*,*20*,*98*,*60*] *89*) **where**
$\Gamma\vdash\ \langle c,s\rangle \Rightarrow\notin T = (\forall\, t.\ \Gamma\vdash\ \langle c,s\rangle \Rightarrow t \longrightarrow t\notin T)$

lemma *final-notinI*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \Longrightarrow t \notin T]\!] \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow\notin T$
by (*simp add*: *final-notin-def*)

lemma *noFaultStuck-Call-body'*: $p \in dom\ \Gamma \Longrightarrow$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) =$
$\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*clarsimp simp add*: *final-notin-def exec-Call-body*)

lemma *noFault-startn*:
assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **and** *t*: $t\neq Fault\ f$
shows $s\neq Fault\ f$
using *execn t* **by** (*induct*) *auto*

lemma *noFault-start*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$ **and** *t*: $t\neq Fault\ f$
shows $s\neq Fault\ f$
using *exec t* **by** (*induct*) *auto*

lemma *noStuck-startn*:
assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **and** *t*: $t\neq Stuck$
shows $s\neq Stuck$
using *execn t* **by** (*induct*) *auto*

lemma *noStuck-start*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$ **and** *t*: $t\neq Stuck$
shows $s\neq Stuck$
using *exec t* **by** (*induct*) *auto*

lemma *noAbrupt-startn*:

assumes *execn*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow t$ **and** *t*: $\forall t'. \ t \neq Abrupt\ t'$
shows $s \neq Abrupt\ s'$
using *execn t* **by** (*induct*) *auto*

lemma *noAbrupt-start*:
assumes *exec*: $\Gamma \vdash \langle c,s \rangle \Rightarrow t$ **and** *t*: $\forall t'. \ t \neq Abrupt\ t'$
shows $s \neq Abrupt\ s'$
using *exec t* **by** (*induct*) *auto*

lemma *noFaultn-startD*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow Normal\ t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-startn*)

lemma *noFaultn-startD'*: $t \neq Fault\ f \Longrightarrow \Gamma \vdash \langle c,s \rangle =n\Rightarrow t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-startn*)

lemma *noFault-startD*: $\Gamma \vdash \langle c,s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-start*)

lemma *noFault-startD'*: $t \neq Fault\ f \Longrightarrow \Gamma \vdash \langle c,s \rangle \Rightarrow t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-start*)

lemma *noStuckn-startD*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow Normal\ t \Longrightarrow s \neq Stuck$
by (*auto dest*: *noStuck-startn*)

lemma *noStuckn-startD'*: $t \neq Stuck \Longrightarrow \Gamma \vdash \langle c,s \rangle =n\Rightarrow t \Longrightarrow s \neq Stuck$
by (*auto dest*: *noStuck-startn*)

lemma *noStuck-startD*: $\Gamma \vdash \langle c,s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Stuck$
by (*auto dest*: *noStuck-start*)

lemma *noStuck-startD'*: $t \neq Stuck \Longrightarrow \Gamma \vdash \langle c,s \rangle \Rightarrow t \Longrightarrow s \neq Stuck$
by (*auto dest*: *noStuck-start*)

lemma *noAbruptn-startD*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow Normal\ t \Longrightarrow s \neq Abrupt\ s'$
by (*auto dest*: *noAbrupt-startn*)

lemma *noAbrupt-startD*: $\Gamma \vdash \langle c,s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Abrupt\ s'$
by (*auto dest*: *noAbrupt-start*)

lemma *noFaultnI*: $[\![\bigwedge t. \ \Gamma \vdash \langle c,s \rangle =n\Rightarrow t \Longrightarrow t \neq Fault\ f]\!] \Longrightarrow \ \Gamma \vdash \langle c,s \rangle =n\Rightarrow \notin \{Fault\ f\}$
by (*simp add*: *nfinal-notin-def*)

lemma *noFaultnI'*:
assumes *contr*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow Fault\ f \Longrightarrow False$
shows $\Gamma \vdash \langle c,s \rangle =n\Rightarrow \notin \{Fault\ f\}$
proof (*rule noFaultnI*)
fix *t* **assume** $\Gamma \vdash \langle c,s \rangle =n\Rightarrow t$
with *contr* **show** $t \neq Fault\ f$

by (*cases* t=*Fault* f) *auto*
qed

lemma *noFaultn-def'*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Fault\ f\} = (\neg\Gamma\vdash\langle c,s\rangle =n\Rightarrow Fault\ f)$
apply *rule*
apply (*fastforce simp add*: *nfinal-notin-def*)
apply (*fastforce intro*: *noFaultnI'*)
done

lemma *noStucknI*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t \Longrightarrow t\neq Stuck]\!] \Longrightarrow \Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Stuck\}$

by (*simp add*: *nfinal-notin-def*)

lemma *noStucknI'*:
assumes *contr*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow Stuck \Longrightarrow False$
shows $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Stuck\}$
proof (*rule noStucknI*)
fix t **assume** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
with *contr* **show** $t \neq Stuck$
by (*cases* t) *auto*
qed

lemma *noStuckn-def'*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Stuck\} = (\neg\Gamma\vdash\langle c,s\rangle =n\Rightarrow Stuck)$
apply *rule*
apply (*fastforce simp add*: *nfinal-notin-def*)
apply (*fastforce intro*: *noStucknI'*)
done

lemma *noFaultI*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq Fault\ f]\!] \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\}$
by (*simp add*: *final-notin-def*)

lemma *noFaultI'*:
assumes *contr*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f\Longrightarrow False$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\}$
proof (*rule noFaultI*)
fix t **assume** $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
with *contr* **show** $t \neq Fault\ f$
by (*cases* t=*Fault* f) *auto*
qed

lemma *noFaultE*:
$[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f]\!] \Longrightarrow P$
by (*auto simp add*: *final-notin-def*)

lemma *noFault-def'*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\} = (\neg\Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f)$
apply *rule*
apply (*fastforce simp add*: *final-notin-def*)
apply (*fastforce intro*: *noFaultI'*)

done

lemma *noStuckI*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq Stuck]\!] \Longrightarrow\ \Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\}$
by (*simp add*: *final-notin-def*)

lemma *noStuckI'*:
assumes *contr*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Stuck \Longrightarrow False$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\}$
proof (*rule noStuckI*)
fix *t* **assume** $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
with *contr* **show** $t \neq Stuck$
by (*cases t*) *auto*
qed

lemma *noStuckE*:
$[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow Stuck]\!] \Longrightarrow P$
by (*auto simp add*: *final-notin-def*)

lemma *noStuck-def'*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\} = (\neg\Gamma\vdash\langle c,s\rangle \Rightarrow Stuck)$
apply *rule*
apply (*fastforce simp add*: *final-notin-def*)
apply (*fastforce intro*: *noStuckI'*)
done

lemma *noFaultn-execD*: $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow t\neq Fault\ f$
by (*simp add*: *nfinal-notin-def*)

lemma *noFault-execD*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow t]\!] \Longrightarrow t\neq Fault\ f$
by (*simp add*: *final-notin-def*)

lemma *noFaultn-exec-startD*: $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow s\neq Fault\ f$
by (*auto simp add*: *nfinal-notin-def dest*: *noFaultn-startD*)

lemma *noFault-exec-startD*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow t]\!] \Longrightarrow s\neq Fault\ f$
by (*auto simp add*: *final-notin-def dest*: *noFault-startD*)

lemma *noStuckn-execD*: $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow t\neq Stuck$
by (*simp add*: *nfinal-notin-def*)

lemma *noStuck-execD*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow t]\!] \Longrightarrow t\neq Stuck$
by (*simp add*: *final-notin-def*)

lemma *noStuckn-exec-startD*: $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow s\neq Stuck$
by (*auto simp add*: *nfinal-notin-def dest*: *noStuckn-startD*)

lemma *noStuck-exec-startD*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow t]\!] \Longrightarrow s\neq Stuck$

by (*auto simp add*: *final-notin-def dest*: *noStuck-startD*)

lemma *noFaultStuckn-execD*:
$[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow$
$t\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\}$
by (*simp add*: *nfinal-notin-def*)

lemma *noFaultStuck-execD*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\};\ \Gamma\vdash\langle c,s\rangle$
$\Rightarrow t]\!]$
$\Longrightarrow t\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\}$
by (*simp add*: *final-notin-def*)

lemma *noFaultStuckn-exec-startD*:
$[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{\mathit{Fault\ True},\ \mathit{Fault\ False},\mathit{Stuck}\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!]$
$\Longrightarrow s\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\}$
by (*auto simp add*: *nfinal-notin-def*)

lemma *noFaultStuck-exec-startD*:
$[\![\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{\mathit{Fault\ True},\ \mathit{Fault\ False},\mathit{Stuck}\};\ \Gamma\vdash\langle c,s\rangle \Rightarrow t]\!]$
$\Longrightarrow s\notin\{\mathit{Fault\ True},\mathit{Fault\ False},\mathit{Stuck}\}$
by (*auto simp add*: *final-notin-def*)

lemma *noStuck-Call*:
assumes *noStuck*: $\Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow\notin\{\mathit{Stuck}\}$
shows $p \in \mathit{dom}\ \Gamma$
proof (*cases* $p \in \mathit{dom}\ \Gamma$)
case *True* **thus** *?thesis* **by** *simp*
next
case *False*
hence $\Gamma\ p = \mathit{None}$ **by** *auto*
hence $\Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Stuck}$
by (*rule exec.CallUndefined*)
with *noStuck* **show** *?thesis*
by (*auto simp add*: *final-notin-def*)
qed

lemma *Guard-noFaultStuckD*:
assumes $\Gamma\vdash\langle \mathit{Guard}\ f\ g\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
assumes $f \notin F$
shows $s \in g$
using *assms*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)

lemma *final-notin-to-finaln*:
assumes *notin*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin T$
shows $\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin T$
proof (*clarsimp simp add*: *nfinal-notin-def*)

fix t **assume** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **and** $t\in T$
with *notin* **show** *False*
by (*auto intro*: *execn-to-exec simp add*: *final-notin-def*)
qed

lemma *noFault-Call-body*:
Γ p=*Some bdy*$\Longrightarrow$
$\Gamma\vdash\langle Call\ p\ ,Normal\ s\rangle \Rightarrow\notin\{Fault\ f\}$ =
$\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin\{Fault\ f\}$
by (*simp add*: *noFault-def′ exec-Call-body*)

lemma *noStuck-Call-body*:
Γ p=*Some bdy*$\Longrightarrow$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$ =
$\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*simp add*: *noStuck-def′ exec-Call-body*)

lemma *exec-final-notin-to-execn*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin T \Longrightarrow \Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin T$
by (*auto simp add*: *final-notin-def nfinal-notin-def dest*: *execn-to-exec*)

lemma *execn-final-notin-to-exec*: $\forall\, n.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin T \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow\notin T$
by (*auto simp add*: *final-notin-def nfinal-notin-def dest*: *exec-to-execn*)

lemma *exec-final-notin-iff-execn*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin T = (\forall\, n.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin T)$
by (*auto intro*: *exec-final-notin-to-execn execn-final-notin-to-exec*)

lemma *Seq-NoFaultStuckD2*:
assumes *noabort*: $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
shows $\forall\, t.\ \Gamma\vdash\langle c1,s\rangle \Rightarrow t \longrightarrow t\notin (\{Stuck\} \cup Fault\ `\ F) \longrightarrow$
$\Gamma\vdash\langle c2,t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
using *noabort*
by (*auto simp add*: *final-notin-def intro*: *exec-Seq′*) **lemma** *Seq-NoFaultStuckD1*:
assumes *noabort*: $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
shows $\Gamma\vdash\langle c1,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
proof (*rule final-notinI*)
fix t
assume *exec-c1*: $\Gamma\vdash\langle c1,s\rangle \Rightarrow t$
show $t \notin \{Stuck\} \cup Fault\ `\ F$
proof
assume $t \in \{Stuck\} \cup Fault\ `\ F$
moreover
{
assume $t = Stuck$
with *exec-c1*
have $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow Stuck$
by (*auto intro*: *exec-Seq′*)
with *noabort* **have** *False*
by (*auto simp add*: *final-notin-def*)
hence *False* **..**

}
moreover
{
assume $t \in \mathit{Fault}\ `\ F$
then obtain f **where**
t: $t=\mathit{Fault}\ f$ **and** f: $f \in F$
by *auto*
from *t exec-c1*
have $\Gamma\vdash\langle \mathit{Seq}\ c1\ c2,s\rangle \Rightarrow \mathit{Fault}\ f$
by (*auto intro*: *exec-Seq′*)
with *noabort f* **have** *False*
by (*auto simp add*: *final-notin-def*)
hence *False* **..**
}
ultimately show *False* **by** *auto*
qed
qed

lemma *Seq-NoFaultStuckD2′*:
assumes *noabort*: $\Gamma\vdash\langle \mathit{Seq}\ c1\ c2,s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F)$
shows $\forall\, t.\ \Gamma\vdash\langle c1,s\rangle \Rightarrow t \longrightarrow t\notin (\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F) \longrightarrow$
$\Gamma\vdash\langle c2,t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F)$
using *noabort*
by (*auto simp add*: *final-notin-def intro*: *exec-Seq′*)

2.3 Lemmas about *sequence*, *flatten* and *Language.normalize*

lemma *execn-sequence-app*: $\bigwedge s\ s'\ t.$
$[\![\Gamma\vdash\langle \mathit{sequence}\ \mathit{Seq}\ \mathit{xs},\mathit{Normal}\ s\rangle =n\Rightarrow s';\ \Gamma\vdash\langle \mathit{sequence}\ \mathit{Seq}\ \mathit{ys},s'\rangle =n\Rightarrow t]\!]$
$\Longrightarrow \Gamma\vdash\langle \mathit{sequence}\ \mathit{Seq}\ (\mathit{xs}@\mathit{ys}),\mathit{Normal}\ s\rangle =n\Rightarrow t$
proof (*induct xs*)
case *Nil*
thus *?case* **by** (*auto elim*: *execn-Normal-elim-cases*)
next
case (*Cons x xs*)
have *exec-x-xs*: $\Gamma\vdash\langle \mathit{sequence}\ \mathit{Seq}\ (x\ \#\ \mathit{xs}),\mathit{Normal}\ s\rangle =n\Rightarrow s'$ **by** *fact*
have *exec-ys*: $\Gamma\vdash\langle \mathit{sequence}\ \mathit{Seq}\ \mathit{ys},s'\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases xs*)
case *Nil*
with *exec-x-xs* **have** $\Gamma\vdash\langle x,\mathit{Normal}\ s\rangle =n\Rightarrow s'$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Nil exec-ys* **show** *?thesis*
by (*cases ys*) (*auto intro*: *execn.intros elim*: *execn-elim-cases*)
next
case *Cons*
with *exec-x-xs*
obtain s'' **where**
exec-x: $\Gamma\vdash\langle x,\mathit{Normal}\ s\rangle =n\Rightarrow s''$ **and**

```
      exec-xs: Γ⊢⟨sequence Seq xs,s''⟩ =n⇒ s'
      by (auto elim: execn-Normal-elim-cases )
    show ?thesis
    proof (cases s'')
      case (Normal s''')
      from Cons.hyps [OF exec-xs [simplified Normal] exec-ys]
      have Γ⊢⟨sequence Seq (xs @ ys),Normal s'''⟩ =n⇒ t .
      with Cons exec-x Normal
      show ?thesis
        by (auto intro: execn.intros)
    next
      case (Abrupt s''')
      with exec-xs have s'=Abrupt s'''
        by (auto dest: execn-Abrupt-end)
      with exec-ys have t=Abrupt s'''
        by (auto dest: execn-Abrupt-end)
      with exec-x Abrupt Cons show ?thesis
        by (auto intro: execn.intros)
    next
      case (Fault f)
      with exec-xs have s'=Fault f
        by (auto dest: execn-Fault-end)
      with exec-ys have t=Fault f
        by (auto dest: execn-Fault-end)
      with exec-x Fault Cons show ?thesis
        by (auto intro: execn.intros)
    next
      case Stuck
      with exec-xs have s'=Stuck
        by (auto dest: execn-Stuck-end)
      with exec-ys have t=Stuck
        by (auto dest: execn-Stuck-end)
      with exec-x Stuck Cons show ?thesis
        by (auto intro: execn.intros)
    qed
  qed
qed

lemma execn-sequence-appD: ⋀s t. Γ⊢⟨sequence Seq (xs @ ys),Normal s⟩ =n⇒ t
⟹
        ∃ s'. Γ⊢⟨sequence Seq xs,Normal s⟩ =n⇒ s' ∧ Γ⊢⟨sequence Seq ys,s'⟩ =n⇒
t
proof (induct xs)
  case Nil
  thus ?case
    by (auto intro: execn.intros)
next
  case (Cons x xs)
  have exec-app: Γ⊢⟨sequence Seq ((x # xs) @ ys),Normal s⟩ =n⇒ t by fact
```

```
  show ?case
  proof (cases xs)
    case Nil
    with exec-app show ?thesis
      by (cases ys) (auto elim: execn-Normal-elim-cases intro: execn-Skip')
  next
    case Cons
    with exec-app obtain s' where
      exec-x: Γ⊢⟨x,Normal s⟩ =n⇒ s' and
      exec-xs-ys: Γ⊢⟨sequence Seq (xs @ ys),s'⟩ =n⇒ t
      by (auto elim: execn-Normal-elim-cases)
    show ?thesis
    proof (cases s')
      case (Normal s'')
      from Cons.hyps [OF exec-xs-ys [simplified Normal]] Normal exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case (Abrupt s'')
      with exec-xs-ys have t=Abrupt s''
        by (auto dest: execn-Abrupt-end)
      with Abrupt exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case (Fault f)
      with exec-xs-ys have t=Fault f
        by (auto dest: execn-Fault-end)
      with Fault exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case Stuck
      with exec-xs-ys have t=Stuck
        by (auto dest: execn-Stuck-end)
      with Stuck exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    qed
  qed
qed

lemma execn-sequence-appE [consumes 1]:
  ⟦Γ⊢⟨sequence Seq (xs @ ys),Normal s⟩ =n⇒ t;
    ⋀s'. ⟦Γ⊢⟨sequence Seq xs,Normal s⟩ =n⇒ s';Γ⊢⟨sequence Seq ys,s'⟩ =n⇒ t⟧
⟹ P
   ⟧ ⟹ P
  by (auto dest: execn-sequence-appD)
```

lemma *execn-to-execn-sequence-flatten*:
  assumes *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
  shows $\Gamma\vdash\langle$*sequence Seq* (*flatten c*),$s\rangle =n\Rightarrow t$
using *exec*
proof *induct*
  case (*Seq c1 c2 n s s′ s′′*) **thus** *?case*
    by (*auto intro*: *execn-sequence-app*)
qed (*auto intro*: *execn.intros*)

lemma *execn-to-execn-normalize*:
  assumes *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
  shows $\Gamma\vdash\langle$*normalize c*,$s\rangle =n\Rightarrow t$
using *exec*
proof *induct*
  case (*Seq c1 c2 n s s′ s′′*) **thus** *?case*
    by (*auto intro*: *execn-to-execn-sequence-flatten execn-sequence-app*)
qed (*auto intro*: *execn.intros*)

lemma *execn-sequence-flatten-to-execn*:
  shows $\bigwedge s\ t.\ \Gamma\vdash\langle$*sequence Seq* (*flatten c*),$s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
proof (*induct c*)
  case (*Seq c1 c2*)
  have *exec-seq*: $\Gamma\vdash\langle$*sequence Seq* (*flatten* (*Seq c1 c2*)),$s\rangle =n\Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases s*)
    case (*Normal s′*)
    with *exec-seq* **obtain** *s′′* **where**
      $\Gamma\vdash\langle$*sequence Seq* (*flatten c1*),*Normal s′*$\rangle =n\Rightarrow s''$ **and**
      $\Gamma\vdash\langle$*sequence Seq* (*flatten c2*),$s''\rangle =n\Rightarrow t$
      by (*auto elim*: *execn-sequence-appE*)
    with *Seq.hyps Normal*
    show *?thesis*
      by (*fastforce intro*: *execn.intros*)
  next
    case *Abrupt*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Abrupt-end*)
  next
    case *Fault*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Fault-end*)
  next
    case *Stuck*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Stuck-end*)
  qed
qed *auto*

lemma *execn-normalize-to-execn*:
  shows $\bigwedge s\ t\ n.\ \Gamma\vdash\langle normalize\ c,s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
proof (*induct c*)
  case *Skip* **thus** *?case* **by** *simp*
next
  case *Basic* **thus** *?case* **by** *simp*
next
  case *Spec* **thus** *?case* **by** *simp*
next
  case (*Seq c1 c2*)
  have $\Gamma\vdash\langle normalize\ (Seq\ c1\ c2),s\rangle =n\Rightarrow t$ **by** *fact*
  hence *exec-norm-seq*:
    $\Gamma\vdash\langle sequence\ Seq\ (flatten\ (normalize\ c1)\ @\ flatten\ (normalize\ c2)),s\rangle =n\Rightarrow t$
    by *simp*
  show *?case*
  proof (*cases s*)
    case (*Normal s'*)
    with *exec-norm-seq* **obtain** s'' **where**
      exec-norm-c1: $\Gamma\vdash\langle sequence\ Seq\ (flatten\ (normalize\ c1)),Normal\ s'\rangle =n\Rightarrow s''$
and
      exec-norm-c2: $\Gamma\vdash\langle sequence\ Seq\ (flatten\ (normalize\ c2)),s''\rangle =n\Rightarrow t$
      by (*auto elim*: *execn-sequence-appE*)
    from *execn-sequence-flatten-to-execn* [*OF exec-norm-c1*]
      execn-sequence-flatten-to-execn [*OF exec-norm-c2*] *Seq.hyps Normal*
    show *?thesis*
      by (*fastforce intro*: *execn.intros*)
  next
    case (*Abrupt s'*)
    with *exec-norm-seq* **have** $t=Abrupt\ s'$
      by (*auto dest*: *execn-Abrupt-end*)
    with *Abrupt* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case (*Fault f*)
    with *exec-norm-seq* **have** $t=Fault\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case *Stuck*
    with *exec-norm-seq* **have** $t=Stuck$
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  qed
next
  case *Cond* **thus** *?case*
    by (*auto intro*: *execn.intros elim!*: *execn-elim-cases*)

```
next
  case (While b c)
  have Γ⊢⟨normalize (While b c),s⟩ =n⇒ t by fact
  hence exec-norm-w: Γ⊢⟨While b (normalize c),s⟩ =n⇒ t
    by simp
  {
    fix s t w
    assume exec-w: Γ⊢⟨w,s⟩ =n⇒ t
    have w=While b (normalize c) ⟹ Γ⊢⟨While b c,s⟩ =n⇒ t
      using exec-w
    proof (induct)
      case (WhileTrue s b' c' n w t)
      from WhileTrue obtain
        s-in-b: s ∈ b and
        exec-c: Γ⊢⟨normalize c,Normal s⟩ =n⇒ w and
        hyp-w: Γ⊢⟨While b c,w⟩ =n⇒ t
        by simp
      from While.hyps [OF exec-c]
      have Γ⊢⟨c,Normal s⟩ =n⇒ w
        by simp
      with hyp-w s-in-b
      have Γ⊢⟨While b c,Normal s⟩ =n⇒ t
        by (auto intro: execn.intros)
      with WhileTrue show ?case by simp
    qed (auto intro: execn.intros)
  }
  from this [OF exec-norm-w]
  show ?case
    by simp
next
  case Call thus ?case by simp
next
  case DynCom thus ?case by (auto intro: execn.intros elim!: execn-elim-cases)
next
  case Guard thus ?case by (auto intro: execn.intros elim!: execn-elim-cases)
next
  case Throw thus ?case by simp
next
  case Catch thus ?case by (fastforce intro: execn.intros elim!: execn-elim-cases)
qed

lemma execn-normalize-iff-execn:
 Γ⊢⟨normalize c,s⟩ =n⇒ t = Γ⊢⟨c,s⟩ =n⇒ t
  by (auto intro: execn-to-execn-normalize execn-normalize-to-execn)

lemma exec-sequence-app:
  assumes exec-xs: Γ⊢⟨sequence Seq xs,Normal s⟩ ⇒ s'
  assumes exec-ys: Γ⊢⟨sequence Seq ys,s'⟩ ⇒ t
  shows Γ⊢⟨sequence Seq (xs@ys),Normal s⟩ ⇒ t
```

proof –
  from *exec-to-execn* [*OF exec-xs*]
  obtain *n* **where**
    execn-xs: $\Gamma\vdash\langle$*sequence Seq xs*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *s'*..
  from *exec-to-execn* [*OF exec-ys*]
  obtain *m* **where**
    execn-ys: $\Gamma\vdash\langle$*sequence Seq ys*,*s'*$\rangle$ =*m*$\Rightarrow$ *t*..
  with *execn-xs* **obtain**
    $\Gamma\vdash\langle$*sequence Seq xs*,*Normal s*$\rangle$ =*max n m*$\Rightarrow$ *s'*
    $\Gamma\vdash\langle$*sequence Seq ys*,*s'*$\rangle$ =*max n m*$\Rightarrow$ *t*
    by (*auto intro*: *execn-mono max.cobounded1 max.cobounded2*)
  from *execn-sequence-app* [*OF this*]
  have $\Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ =*max n m*$\Rightarrow$ *t* .
  thus *?thesis*
    by (*rule execn-to-exec*)
qed

lemma *exec-sequence-appD*:
  assumes *exec-xs-ys*: $\Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ $\Rightarrow$ *t*
  shows $\exists$ *s'*. $\Gamma\vdash\langle$*sequence Seq xs*,*Normal s*$\rangle$ $\Rightarrow$ *s'* $\wedge$ $\Gamma\vdash\langle$*sequence Seq ys*,*s'*$\rangle$ $\Rightarrow$ *t*
proof –
  from *exec-to-execn* [*OF exec-xs-ys*]
  obtain *n* **where** $\Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*..
  thus *?thesis*
    by (*cases rule*: *execn-sequence-appE*) (*auto intro*: *execn-to-exec*)
qed

lemma *exec-sequence-appE* [*consumes 1*]:
  $[\![$$\Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ $\Rightarrow$ *t*;
   $\bigwedge$*s'*. $[\![$$\Gamma\vdash\langle$*sequence Seq xs*,*Normal s*$\rangle$ $\Rightarrow$ *s'*;$\Gamma\vdash\langle$*sequence Seq ys*,*s'*$\rangle$ $\Rightarrow$ *t*$]\!]$ $\Longrightarrow$ *P*
   $]\!]$ $\Longrightarrow$ *P*
  by (*auto dest*: *exec-sequence-appD*)

lemma *exec-to-exec-sequence-flatten*:
  assumes *exec*: $\Gamma\vdash\langle$*c*,*s*$\rangle$ $\Rightarrow$ *t*
  shows $\Gamma\vdash\langle$*sequence Seq* (*flatten c*),*s*$\rangle$ $\Rightarrow$ *t*
proof –
  from *exec-to-execn* [*OF exec*]
  obtain *n* **where** $\Gamma\vdash\langle$*c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*..
  from *execn-to-execn-sequence-flatten* [*OF this*]
  show *?thesis*
    by (*rule execn-to-exec*)
qed

lemma *exec-sequence-flatten-to-exec*:
  assumes *exec-seq*: $\Gamma\vdash\langle$*sequence Seq* (*flatten c*),*s*$\rangle$ $\Rightarrow$ *t*
  shows $\Gamma\vdash\langle$*c*,*s*$\rangle$ $\Rightarrow$ *t*
proof –

from *exec-to-execn* [*OF exec-seq*]
obtain n **where** $\Gamma\vdash\langle$*sequence Seq* (*flatten c*),$s\rangle$ $=n\Rightarrow$ t..
from *execn-sequence-flatten-to-execn* [*OF this*]
show *?thesis*
by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-normalize*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
shows $\Gamma\vdash\langle$*normalize* $c,s\rangle \Rightarrow t$
proof –
from *exec-to-execn* [*OF exec*] **obtain** n **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$..
hence $\Gamma\vdash\langle$*normalize* $c,s\rangle =n\Rightarrow t$
by (*rule execn-to-execn-normalize*)
thus *?thesis*
by (*rule execn-to-exec*)
qed

lemma *exec-normalize-to-exec*:
assumes *exec*: $\Gamma\vdash\langle$*normalize* $c,s\rangle \Rightarrow t$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
proof –
from *exec-to-execn* [*OF exec*] **obtain** n **where** $\Gamma\vdash\langle$*normalize* $c,s\rangle =n\Rightarrow t$..
hence $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
by (*rule execn-normalize-to-execn*)
thus *?thesis*
by (*rule execn-to-exec*)
qed

lemma *exec-normalize-iff-exec*:
$\Gamma\vdash\langle$*normalize* $c,s\rangle \Rightarrow t = \Gamma\vdash\langle c,s\rangle \Rightarrow t$
by (*auto intro*: *exec-to-exec-normalize exec-normalize-to-exec*)

2.4 Lemmas about $c_1 \subseteq_g c_2$

lemma *execn-to-execn-subseteq-guards*: $\bigwedge c\ s\ t\ n.\ [\![c \subseteq_g c'; \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!]$
$\Longrightarrow \exists\, t'.\ \Gamma\vdash\langle c',s\rangle =n\Rightarrow t' \wedge$
(*isFault* $t \longrightarrow$ *isFault* t') $\wedge$ ($\neg$ *isFault* $t' \longrightarrow t'=t$)
proof (*induct* c')
case *Skip* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case *Basic* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case *Spec* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case (*Seq c1' c2'*)

have $c \subseteq_g$ *Seq c1′ c2′* **by** *fact*
from *subseteq-guards-Seq* [*OF this*]
obtain *c1 c2* **where**
  c: c = *Seq c1 c2* **and**
  c1-c1′: *c1* $\subseteq_g$ *c1′* **and**
  c2-c2′: *c2* $\subseteq_g$ *c2′*
  by *blast*
have *exec*: $\Gamma\vdash\langle c,s\rangle$ $=n\Rightarrow$ *t* **by** *fact*
with *c* **obtain** *w* **where**
  exec-c1: $\Gamma\vdash\langle c1,s\rangle$ $=n\Rightarrow$ *w* **and**
  exec-c2: $\Gamma\vdash\langle c2,w\rangle$ $=n\Rightarrow$ *t*
  by (*auto elim*: *execn-elim-cases*)
from *exec-c1 Seq.hyps c1-c1′*
obtain *w′* **where**
  exec-c1′: $\Gamma\vdash\langle c1',s\rangle$ $=n\Rightarrow$ *w′* **and**
  w-Fault: *isFault w* $\longrightarrow$ *isFault w′* **and**
  w′-noFault: $\neg$ *isFault w′* $\longrightarrow$ *w′=w*
  by *blast*
show *?case*
proof (*cases s*)
  case (*Fault f*)
  with *exec* **have** *t=Fault f*
    by (*auto dest*: *execn-Fault-end*)
  with *Fault* **show** *?thesis*
    by *auto*
next
  case *Stuck*
  with *exec* **have** *t=Stuck*
    by (*auto dest*: *execn-Stuck-end*)
  with *Stuck* **show** *?thesis*
    by *auto*
next
  case (*Abrupt s′*)
  with *exec* **have** *t=Abrupt s′*
    by (*auto dest*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis*
    by *auto*
next
  case (*Normal s′*)
  show *?thesis*
  proof (*cases isFault w*)
    case *True*
    then obtain *f* **where** *w′*: *w=Fault f*..
    moreover with *exec-c2*
    have *t*: *t=Fault f*
      by (*auto dest*: *execn-Fault-end*)
    ultimately show *?thesis*
      using *Normal w-Fault exec-c1′*
      by (*fastforce intro*: *execn.intros elim*: *isFaultE*)

```
  next
    case False
    note noFault-w = this
    show ?thesis
    proof (cases isFault w')
      case True
      then obtain f' where w': w'=Fault f'..
      with Normal exec-c1'
      have exec: Γ⊢⟨Seq c1' c2',s⟩ =n⇒ Fault f'
        by (auto intro: execn.intros)
      then show ?thesis
        by auto
    next
      case False
      with w'-noFault have w': w'=w by simp
      from Seq.hyps exec-c2 c2-c2'
      obtain t' where
        Γ⊢⟨c2',w⟩ =n⇒ t' and
        isFault t ⟶ isFault t' and
        ¬ isFault t' ⟶ t'=t
        by blast
      with Normal exec-c1' w'
      show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
 qed
next
 case (Cond b c1' c2')
 have exec: Γ⊢⟨c,s⟩ =n⇒ t by fact
 have c ⊆g Cond b c1' c2' by fact
 from subseteq-guards-Cond [OF this]
 obtain c1 c2 where
   c: c = Cond b c1 c2 and
   c1-c1': c1 ⊆g c1' and
   c2-c2': c2 ⊆g c2'
   by blast
 show ?case
 proof (cases s)
   case (Fault f)
   with exec have t=Fault f
     by (auto dest: execn-Fault-end)
   with Fault show ?thesis
     by auto
 next
   case Stuck
   with exec have t=Stuck
     by (auto dest: execn-Stuck-end)
   with Stuck show ?thesis
```

```
      by auto
  next
    case (Abrupt s')
    with exec have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
    from exec [simplified c Normal]
    show ?thesis
    proof (cases)
      assume s'-in-b: s' ∈ b
      assume Γ⊢⟨c1,Normal s'⟩ =n⇒ t
      with c1-c1' Normal Cond.hyps obtain t' where
        Γ⊢⟨c1',Normal s'⟩ =n⇒ t'
        isFault t ⟶ isFault t'
        ¬ isFault t' ⟶ t' = t
        by blast
      with s'-in-b Normal show ?thesis
        by (fastforce intro: execn.intros)
    next
      assume s'-notin-b: s' ∉ b
      assume Γ⊢⟨c2,Normal s'⟩ =n⇒ t
      with c2-c2' Normal Cond.hyps obtain t' where
        Γ⊢⟨c2',Normal s'⟩ =n⇒ t'
        isFault t ⟶ isFault t'
        ¬ isFault t' ⟶ t' = t
        by blast
      with s'-notin-b Normal show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
next
  case (While b c')
  have exec: Γ⊢⟨c,s⟩ =n⇒ t by fact
  have c ⊆g While b c' by fact
  from subseteq-guards-While [OF this]
  obtain c'' where
    c: c = While b c'' and
    c''-c': c'' ⊆g c'
    by blast
  {
    fix c r w
    assume exec: Γ⊢⟨c,r⟩ =n⇒ w
    assume c: c=While b c''
    have ∃ w'. Γ⊢⟨While b c',r⟩ =n⇒ w' ∧
              (isFault w ⟶ isFault w') ∧ (¬ isFault w' ⟶ w'=w)
    using exec c
```

proof (*induct*)
case (*WhileTrue r b' ca n u w*)
have *eqs*: *While b' ca = While b c''* **by** *fact*
from *WhileTrue* **have** *r-in-b*: $r \in b$ **by** *simp*
from *WhileTrue* **have** *exec-c''*: $\Gamma\vdash\langle c'',Normal\ r\rangle =n\Rightarrow u$ **by** *simp*
from *While.hyps* [*OF c''-c' exec-c''*] **obtain** *u'* **where**
exec-c': $\Gamma\vdash\langle c',Normal\ r\rangle =n\Rightarrow u'$ **and**
u-Fault: *isFault u* $\longrightarrow$ *isFault u'* **and**
u'-noFault: $\neg$ *isFault u'* $\longrightarrow$ $u' = u$
by *blast*
from *WhileTrue* **obtain** *w'* **where**
exec-w: $\Gamma\vdash\langle While\ b\ c',u\rangle =n\Rightarrow w'$ **and**
w-Fault: *isFault w* $\longrightarrow$ *isFault w'* **and**
w'-noFault: $\neg$ *isFault w'* $\longrightarrow$ $w' = w$
by *blast*
show *?case*
proof (*cases isFault u'*)
case *True*
with *exec-c' r-in-b*
show *?thesis*
by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
case *False*
with *exec-c' r-in-b u'-noFault exec-w w-Fault w'-noFault*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
next
case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
qed *auto*
}
from *this* [*OF exec c*]
show *?case* **.**
next
case *Call* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case (*DynCom C'*)
have *exec*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **by** *fact*
have $c \subseteq_g$ *DynCom C'* **by** *fact*
from *subseteq-guards-DynCom* [*OF this*] **obtain** *C* **where**
c: *c = DynCom C* **and**
C-C': $\forall s.\ C\ s \subseteq_g C'\ s$
by *blast*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)

with *Fault* **show** *?thesis*
  by *auto*
next
  case *Stuck*
  with *exec* **have** $t{=}Stuck$
    by (*auto dest*: *execn-Stuck-end*)
  with *Stuck* **show** *?thesis*
    by *auto*
next
  case (*Abrupt* s')
  with *exec* **have** $t{=}Abrupt\ s'$
    by (*auto dest*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis*
    by *auto*
next
  case (*Normal* s')
  from *exec* [*simplified c Normal*]
  have $\Gamma\vdash\langle C\ s',Normal\ s'\rangle\ =n\Rightarrow\ t$
    by *cases*
  from *DynCom.hyps* C-C' [*rule-format*] *this* **obtain** t' **where**
    $\Gamma\vdash\langle C'\ s',Normal\ s'\rangle\ =n\Rightarrow\ t'$
    $isFault\ t \longrightarrow isFault\ t'$
    $\neg\ isFault\ t' \longrightarrow t' = t$
    by *blast*
  with *Normal* **show** *?thesis*
    by (*fastforce intro*: *execn.intros*)
qed
next
  case (*Guard* $f'\ g'\ c'$)
  have *exec*: $\Gamma\vdash\langle c,s\rangle\ =n\Rightarrow\ t$ **by** *fact*
  have $c \subseteq_g Guard\ f'\ g'\ c'$ **by** *fact*
  hence *subset-cases*: $(c \subseteq_g c') \vee (\exists\, c''.\ c = Guard\ f'\ g'\ c'' \wedge (c'' \subseteq_g c'))$
    by (*rule subseteq-guards-Guard*)
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec* **have** $t{=}Fault\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec* **have** $t{=}Stuck$
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case (*Abrupt* s')
    with *exec* **have** $t{=}Abrupt\ s'$

by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s')
from *subset-cases* **show** *?thesis*
proof
assume *c-c'*: $c \subseteq_g c'$
from *Guard.hyps* [*OF this exec*] *Normal* **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c',$*Normal* $s'\rangle$ $=n\Rightarrow$ t' **and**
t-Fault: *isFault* $t \longrightarrow$ *isFault* t' **and**
t-noFault: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
with *Normal*
show *?thesis*
by (*cases* $s' \in g'$) (*fastforce intro*: *execn.intros*)+
next
assume $\exists\, c''.\ c =$ *Guard* $f'\ g'\ c'' \wedge (c'' \subseteq_g c')$
then obtain c'' **where**
c: $c =$ *Guard* $f'\ g'\ c''$ **and**
c''-c': $c'' \subseteq_g c'$
by *blast*
from *c exec Normal*
have *exec-Guard'*: $\Gamma\vdash\langle$*Guard* $f'\ g'\ c'',$*Normal* $s'\rangle$ $=n\Rightarrow$ t
by *simp*
thus *?thesis*
proof (*cases*)
assume *s'-in-g'*: $s' \in g'$
assume *exec-c''*: $\Gamma\vdash\langle c'',$*Normal* $s'\rangle$ $=n\Rightarrow$ t
from *Guard.hyps* [*OF c''-c' exec-c''*] **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c',$*Normal* $s'\rangle$ $=n\Rightarrow$ t' **and**
t-Fault: *isFault* $t \longrightarrow$ *isFault* t' **and**
t-noFault: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
with *Normal s'-in-g'*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
next
assume $s' \notin g'$ t=*Fault* f'
with *Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
qed
next
case *Throw* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD intro*: *execn.intros*
elim: *execn-elim-cases*)
next

```
case (Catch c1' c2')
have c ⊆g Catch c1' c2' by fact
from subseteq-guards-Catch [OF this]
obtain c1 c2 where
  c: c = Catch c1 c2 and
  c1-c1': c1 ⊆g c1' and
  c2-c2': c2 ⊆g c2'
  by blast
have exec: Γ⊢⟨c,s⟩ =n⇒ t by fact
show ?case
proof (cases s)
  case (Fault f)
  with exec have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
  case Stuck
  with exec have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  from exec [simplified c Normal]
  show ?thesis
  proof (cases)
    fix w
    assume exec-c1: Γ⊢⟨c1,Normal s'⟩ =n⇒ Abrupt w
    assume exec-c2: Γ⊢⟨c2,Normal w⟩ =n⇒ t
    from Normal exec-c1 c1-c1' Catch.hyps obtain w' where
      exec-c1': Γ⊢⟨c1',Normal s'⟩ =n⇒ w' and
      w'-noFault: ¬ isFault w' ⟶ w' = Abrupt w
      by blast
    show ?thesis
    proof (cases isFault w')
      case True
      with exec-c1' Normal show ?thesis
        by (fastforce intro: execn.intros elim: isFaultE)
    next
      case False
      with w'-noFault have w': w'=Abrupt w by simp
      from Normal exec-c2 c2-c2' Catch.hyps obtain t' where
```

$\Gamma\vdash\langle c2',Normal\ w\rangle =n\Rightarrow t'$
isFault t $\longrightarrow$ *isFault t'*
$\neg$ *isFault t'* $\longrightarrow$ $t' = t$
by *blast*
with *exec-c1' w' Normal*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
next
assume *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s'\rangle =n\Rightarrow t$
assume *t*: $\neg$ *isAbr t*
from *Normal exec-c1 c1-c1' Catch.hyps* **obtain** *t'* **where**
exec-c1': $\Gamma\vdash\langle c1',Normal\ s'\rangle =n\Rightarrow t'$ **and**
t-Fault: *isFault t* $\longrightarrow$ *isFault t'* **and**
t'-noFault: $\neg$ *isFault t'* $\longrightarrow$ $t' = t$
by *blast*
show *?thesis*
proof (*cases isFault t'*)
case *True*
with *exec-c1' Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
case *False*
with *exec-c1' Normal t-Fault t'-noFault t*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
qed
qed

lemma *exec-to-exec-subseteq-guards*:
assumes *c-c'*: $c \subseteq_g c'$
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
shows $\exists t'.\ \Gamma\vdash\langle c',s\rangle \Rightarrow t' \wedge$
(*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$ ($\neg$ *isFault t'* $\longrightarrow$ $t'=t$)
proof −
from *exec-to-execn* [*OF exec*] **obtain** *n* **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **..**
from *execn-to-execn-subseteq-guards* [*OF c-c' this*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

2.5 Lemmas about *merge-guards*

theorem *execn-to-execn-merge-guards*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
shows $\Gamma\vdash\langle$*merge-guards c*,$s\rangle =n\Rightarrow t$

```
using exec-c
proof (induct)
  case (Guard s g c n t f)
  have s-in-g: s ∈ g  by fact
  have exec-merge-c: Γ⊢⟨merge-guards c,Normal s⟩ =n⇒ t by fact
  show ?case
  proof (cases ∃f′ g′ c′. merge-guards c = Guard f′ g′ c′)
    case False
    with exec-merge-c s-in-g
    show ?thesis
      by (cases merge-guards c) (auto intro: execn.intros simp add: Let-def)
  next
    case True
    then obtain f′ g′ c′ where
      merge-guards-c: merge-guards c = Guard f′ g′ c′
      by iprover
    show ?thesis
    proof (cases f=f′)
      case False
      from exec-merge-c s-in-g merge-guards-c False show ?thesis
        by (auto intro: execn.intros simp add: Let-def)
    next
      case True
      from exec-merge-c s-in-g merge-guards-c True show ?thesis
        by (fastforce intro: execn.intros elim: execn.cases)
    qed
  qed
next
  case (GuardFault s g f c n)
  have s-notin-g: s ∉ g  by fact
  show ?case
  proof (cases ∃f′ g′ c′. merge-guards c = Guard f′ g′ c′)
    case False
    with s-notin-g
    show ?thesis
      by (cases merge-guards c) (auto intro: execn.intros simp add: Let-def)
  next
    case True
    then obtain f′ g′ c′ where
      merge-guards-c: merge-guards c = Guard f′ g′ c′
      by iprover
    show ?thesis
    proof (cases f=f′)
      case False
      from s-notin-g merge-guards-c False show ?thesis
        by (auto intro: execn.intros simp add: Let-def)
    next
      case True
      from  s-notin-g merge-guards-c True show ?thesis
```

by (*fastforce intro*: *execn.intros*)
qed
qed
qed (*fastforce intro*: *execn.intros*)+

lemma *execn-merge-guards-to-execn-Normal*:
$\bigwedge s\ n\ t.\ \Gamma\vdash\langle merge\text{-}guards\ c,Normal\ s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
proof (*induct c*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case (*Seq c1 c2*)
have $\Gamma\vdash\langle merge\text{-}guards\ (Seq\ c1\ c2),Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
hence *exec-merge*: $\Gamma\vdash\langle Seq\ (merge\text{-}guards\ c1)\ (merge\text{-}guards\ c2),Normal\ s\rangle =n\Rightarrow$
t
by *simp*
then obtain s' **where**
exec-merge-c1: $\Gamma\vdash\langle merge\text{-}guards\ c1,Normal\ s\rangle =n\Rightarrow s'$ **and**
exec-merge-c2: $\Gamma\vdash\langle merge\text{-}guards\ c2,s'\rangle =n\Rightarrow t$
by *cases*
from *exec-merge-c1*
have *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow s'$
by (*rule Seq.hyps*)
show *?case*
proof (*cases* s')
case (*Normal* s'')
with *exec-merge-c2*
have $\Gamma\vdash\langle c2,s'\rangle =n\Rightarrow t$
by (*auto intro*: *Seq.hyps*)
with *exec-c1* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Abrupt* s'')
with *exec-merge-c2* **have** $t{=}Abrupt\ s''$
by (*auto dest*: *execn-Abrupt-end*)
with *exec-c1 Abrupt*
show *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Fault f*)
with *exec-merge-c2* **have** $t{=}Fault\ f$
by (*auto dest*: *execn-Fault-end*)
with *exec-c1 Fault*
show *?thesis*
by (*auto intro*: *execn.intros*)
next

```
    case Stuck
    with exec-merge-c2 have t=Stuck
      by (auto dest: execn-Stuck-end)
    with exec-c1 Stuck
    show ?thesis
      by (auto intro: execn.intros)
  qed
next
  case Cond thus ?case
    by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
  case (While b c)
  {
    fix c' r w
    assume exec-c': Γ⊢⟨c',r⟩ =n⇒ w
    assume c': c'=While b (merge-guards c)
    have Γ⊢⟨While b c,r⟩ =n⇒ w
      using exec-c' c'
    proof (induct)
      case (WhileTrue r b' c'' n u w)
      have eqs: While b' c'' = While b (merge-guards c) by fact
      from WhileTrue
      have r-in-b: r ∈ b
        by simp
      from WhileTrue While.hyps have exec-c: Γ⊢⟨c,Normal r⟩ =n⇒ u
        by simp
      from WhileTrue have exec-w: Γ⊢⟨While b c,u⟩ =n⇒ w
        by simp
      from r-in-b exec-c exec-w
      show ?case
        by (rule execn.WhileTrue)
    next
      case WhileFalse thus ?case by (auto intro: execn.WhileFalse)
    qed auto
  }
  with While.prems show ?case
    by (auto)
next
  case Call thus ?case by simp
next
  case DynCom thus ?case
    by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
  case (Guard f g c)
  have exec-merge: Γ⊢⟨merge-guards (Guard f g c),Normal s⟩ =n⇒ t by fact
  show ?case
  proof (cases s ∈ g)
    case False
    with exec-merge have t=Fault f
```

```
    by (auto split: com.splits if-split-asm elim: execn-Normal-elim-cases
      simp add: Let-def is-Guard-def)
  with False show ?thesis
    by (auto intro: execn.intros)
next
  case True
  note s-in-g = this
  show ?thesis
  proof (cases ∃f' g' c'. merge-guards c = Guard f' g' c')
    case False
    then
    have merge-guards (Guard f g c) = Guard f g (merge-guards c)
      by (cases merge-guards c) (auto simp add: Let-def)
    with exec-merge s-in-g
    obtain Γ⊢⟨merge-guards c,Normal s⟩ =n⇒ t
      by (auto elim: execn-Normal-elim-cases)
    from Guard.hyps [OF this] s-in-g
    show ?thesis
      by (auto intro: execn.intros)
  next
    case True
    then obtain f' g' c' where
      merge-guards-c: merge-guards c = Guard f' g' c'
      by iprover
    show ?thesis
    proof (cases f=f')
      case False
      with merge-guards-c
      have merge-guards (Guard f g c) = Guard f g (merge-guards c)
        by (simp add: Let-def)
      with exec-merge s-in-g
      obtain Γ⊢⟨merge-guards c,Normal s⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      from Guard.hyps [OF this] s-in-g
      show ?thesis
        by (auto intro: execn.intros)
    next
      case True
      note f-eq-f' = this
      with merge-guards-c have
        merge-guards-Guard: merge-guards (Guard f g c) = Guard f (g ∩ g') c'
        by simp
      show ?thesis
      proof (cases s ∈ g')
        case True
        with exec-merge merge-guards-Guard merge-guards-c s-in-g
        have Γ⊢⟨merge-guards c,Normal s⟩ =n⇒ t
          by (auto intro: execn.intros elim: execn-Normal-elim-cases)
        with Guard.hyps [OF this] s-in-g
```

```
          show ?thesis
            by (auto intro: execn.intros)
        next
          case False
          with exec-merge merge-guards-Guard
          have t=Fault f
            by (auto elim: execn-Normal-elim-cases)
          with merge-guards-c f-eq-f' False
          have Γ⊢⟨merge-guards c,Normal s⟩ =n⇒ t
            by (auto intro: execn.intros)
          from Guard.hyps [OF this] s-in-g
          show ?thesis
            by (auto intro: execn.intros)
        qed
      qed
    qed
  qed
next
  case Throw thus ?case by simp
next
  case (Catch c1 c2)
  have Γ⊢⟨merge-guards (Catch c1 c2),Normal s⟩ =n⇒ t  by fact
   hence Γ⊢⟨Catch (merge-guards c1) (merge-guards c2),Normal s⟩ =n⇒ t by
simp
  thus ?case
    by cases (auto intro: execn.intros Catch.hyps)
qed

theorem execn-merge-guards-to-execn:
  Γ⊢⟨merge-guards c,s⟩ =n⇒ t ⟹ Γ⊢⟨c, s⟩ =n⇒ t
apply (cases s)
apply    (fastforce intro: execn-merge-guards-to-execn-Normal)
apply   (fastforce dest: execn-Abrupt-end)
apply  (fastforce dest: execn-Fault-end)
apply (fastforce dest: execn-Stuck-end)
done

corollary execn-iff-execn-merge-guards:
 Γ⊢⟨c, s⟩ =n⇒ t = Γ⊢⟨merge-guards c,s⟩ =n⇒ t
  by (blast intro: execn-merge-guards-to-execn execn-to-execn-merge-guards)

theorem exec-iff-exec-merge-guards:
 Γ⊢⟨c, s⟩ ⇒ t = Γ⊢⟨merge-guards c,s⟩ ⇒ t
  by (blast dest: exec-to-execn intro: execn-to-exec
           intro: execn-to-execn-merge-guards
                  execn-merge-guards-to-execn)

corollary exec-to-exec-merge-guards:
 Γ⊢⟨c, s⟩ ⇒ t ⟹ Γ⊢⟨merge-guards c,s⟩ ⇒ t
```

by (*rule iffD1* [*OF exec-iff-exec-merge-guards*])

corollary *exec-merge-guards-to-exec*:
$\Gamma\vdash\langle$*merge-guards c*,*s*$\rangle \Rightarrow t \Longrightarrow \Gamma\vdash\langle c,\ s\rangle \Rightarrow t$
by (*rule iffD2* [*OF exec-iff-exec-merge-guards*])

2.6 Lemmas about *mark-guards*

lemma *execn-to-execn-mark-guards*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma\vdash\langle$*mark-guards f c*,*s*$\rangle =n\Rightarrow t$
using *exec-c t-not-Fault* [*simplified not-isFault-iff*]
by (*induct*) (*auto intro*: *execn.intros dest*: *noFaultn-startD'*)

lemma *execn-to-execn-mark-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
shows $\bigwedge f.\ [\![t=Fault\ f]\!] \Longrightarrow \exists f'.\ \Gamma\vdash\langle$*mark-guards x c*,*s*$\rangle =n\Rightarrow$ *Fault f'*
using *exec-c*
proof (*induct*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Guard* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case *SpecStuck* **thus** *?case* **by** *auto*
next
case (*Seq c1 s n w c2 t*)
have *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow w$ **by** *fact*
have *exec-c2*: $\Gamma\vdash\langle c2,w\rangle =n\Rightarrow t$ **by** *fact*
have *t*: *t=Fault f* **by** *fact*
show *?case*
proof (*cases w*)
case (*Fault f'*)
with *exec-c2 t* **have** *f'=f*
by (*auto dest*: *execn-Fault-end*)
with *Fault Seq.hyps* **obtain** *f''* **where**
$\Gamma\vdash\langle$*mark-guards x c1*,*Normal s*$\rangle =n\Rightarrow$ *Fault f''*
by *auto*
moreover have $\Gamma\vdash\langle$*mark-guards x c2*,*Fault f''*$\rangle =n\Rightarrow$ *Fault f''*
by *auto*
ultimately show *?thesis*

```
      by (auto intro: execn.intros)
  next
    case (Normal s')
    with execn-to-execn-mark-guards [OF exec-c1]
    have exec-mark-c1: Γ⊢⟨mark-guards x c1,Normal s⟩ =n⇒ w
      by simp
    with Seq.hyps t obtain f' where
      Γ⊢⟨mark-guards x c2,w⟩ =n⇒ Fault f'
      by blast
    with exec-mark-c1 show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-mark-guards [OF exec-c1]
    have exec-mark-c1: Γ⊢⟨mark-guards x c1,Normal s⟩ =n⇒ w
      by simp
    with Seq.hyps t obtain f' where
      Γ⊢⟨mark-guards x c2,w⟩ =n⇒ Fault f'
      by (auto intro: execn.intros)
    with exec-mark-c1 show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-c2 have t=Stuck
      by (auto dest: execn-Stuck-end)
    with t show ?thesis by simp
  qed
next
  case CondTrue thus ?case by (fastforce intro: execn.intros)
next
  case CondFalse thus ?case by (fastforce intro: execn.intros)
next
  case (WhileTrue s b c n w t)
  have exec-c: Γ⊢⟨c,Normal s⟩ =n⇒ w by fact
  have exec-w: Γ⊢⟨While b c,w⟩ =n⇒ t by fact
  have t: t = Fault f by fact
  have s-in-b: s ∈ b by fact
  show ?case
  proof (cases w)
    case (Fault f')
    with exec-w t have f'=f
      by (auto dest: execn-Fault-end)
    with Fault WhileTrue.hyps obtain f'' where
      Γ⊢⟨mark-guards x c,Normal s⟩ =n⇒ Fault f''
      by auto
    moreover have Γ⊢⟨mark-guards x (While b c),Fault f''⟩ =n⇒ Fault f''
      by auto
    ultimately show ?thesis
      using s-in-b by (auto intro: execn.intros)
```

next
  case (*Normal s'*)
  with *execn-to-execn-mark-guards* [*OF exec-c*]
  have *exec-mark-c*: $\Gamma\vdash\langle$*mark-guards x c*,*Normal s*$\rangle$ $=n\Rightarrow$ *w*
    by *simp*
  with *WhileTrue.hyps t* **obtain** *f'* **where**
    $\Gamma\vdash\langle$*mark-guards x* (*While b c*),*w*$\rangle$ $=n\Rightarrow$ *Fault f'*
    by *blast*
  with *exec-mark-c s-in-b* **show** *?thesis*
    by (*auto intro*: *execn.intros*)
next
  case (*Abrupt s'*)
  with *execn-to-execn-mark-guards* [*OF exec-c*]
  have *exec-mark-c*: $\Gamma\vdash\langle$*mark-guards x c*,*Normal s*$\rangle$ $=n\Rightarrow$ *w*
    by *simp*
  with *WhileTrue.hyps t* **obtain** *f'* **where**
    $\Gamma\vdash\langle$*mark-guards x* (*While b c*),*w*$\rangle$ $=n\Rightarrow$ *Fault f'*
    by (*auto intro*: *execn.intros*)
  with *exec-mark-c s-in-b* **show** *?thesis*
    by (*auto intro*: *execn.intros*)
next
  case *Stuck*
  with *exec-w* **have** *t=Stuck*
    by (*auto dest*: *execn-Stuck-end*)
  with *t* **show** *?thesis* **by** *simp*
qed
next
  case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *Call* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *CallUndefined* **thus** *?case* **by** *simp*
next
  case *StuckProp* **thus** *?case* **by** *simp*
next
  case *DynCom* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *Throw* **thus** *?case* **by** *simp*
next
  case *AbruptProp* **thus** *?case* **by** *simp*
next
  case (*CatchMatch c1 s n w c2 t*)
  have *exec-c1*: $\Gamma\vdash\langle$*c1*,*Normal s*$\rangle$ $=n\Rightarrow$ *Abrupt w* **by** *fact*
  have *exec-c2*: $\Gamma\vdash\langle$*c2*,*Normal w*$\rangle$ $=n\Rightarrow$ *t* **by** *fact*
  have *t*: *t* = *Fault f* **by** *fact*
  from *execn-to-execn-mark-guards* [*OF exec-c1*]
  have *exec-mark-c1*: $\Gamma\vdash\langle$*mark-guards x c1*,*Normal s*$\rangle$ $=n\Rightarrow$ *Abrupt w*
    by *simp*
  with *CatchMatch.hyps t* **obtain** *f'* **where**

$\Gamma\vdash\langle$*mark-guards x c2*,*Normal w*$\rangle$ =*n*$\Rightarrow$ *Fault f'*
by *blast*
with *exec-mark-c1* **show** *?case*
by (*auto intro*: *execn.intros*)
next
case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
qed

lemma *execn-mark-guards-to-execn*:
$\bigwedge$*s n t*. $\Gamma\vdash\langle$*mark-guards f c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
$\Longrightarrow$ $\exists$ *t'*. $\Gamma\vdash\langle$*c*,*s*$\rangle$ =*n*$\Rightarrow$ *t'* $\wedge$
(*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$
(*t'* = *Fault f* $\longrightarrow$ *t'*=*t*) $\wedge$
(*isFault t'* $\longrightarrow$ *isFault t*) $\wedge$
($\neg$ *isFault t'* $\longrightarrow$ *t'*=*t*)
proof (*induct c*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case (*Seq c1 c2 s n t*)
have *exec-mark*: $\Gamma\vdash\langle$*mark-guards f* (*Seq c1 c2*),*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
then obtain *w* **where**
exec-mark-c1: $\Gamma\vdash\langle$*mark-guards f c1*,*s*$\rangle$ =*n*$\Rightarrow$ *w* **and**
exec-mark-c2: $\Gamma\vdash\langle$*mark-guards f c2*,*w*$\rangle$ =*n*$\Rightarrow$ *t*
by (*auto elim*: *execn-elim-cases*)
from *Seq.hyps exec-mark-c1*
obtain *w'* **where**
exec-c1: $\Gamma\vdash\langle$*c1*,*s*$\rangle$ =*n*$\Rightarrow$ *w'* **and**
w-Fault: *isFault w* $\longrightarrow$ *isFault w'* **and**
w'-Fault-f: *w'* = *Fault f* $\longrightarrow$ *w'*=*w* **and**
w'-Fault: *isFault w'* $\longrightarrow$ *isFault w* **and**
w'-noFault: $\neg$ *isFault w'* $\longrightarrow$ *w'*=*w*
by *blast*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-mark* **have** *t*=*Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-mark* **have** *t*=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*

next
  case (*Abrupt s'*)
  with *exec-mark* **have** *t=Abrupt s'*
    by (*auto dest*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis*
    by *auto*
next
  case (*Normal s'*)
  show *?thesis*
  proof (*cases isFault w*)
    case *True*
    then obtain *f* **where** *w'*: *w=Fault f*..
    moreover with *exec-mark-c2*
    have *t*: *t=Fault f*
      by (*auto dest*: *execn-Fault-end*)
    ultimately show *?thesis*
      using *Normal w-Fault w'-Fault-f exec-c1*
      by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
  next
    case *False*
    note *noFault-w = this*
    show *?thesis*
    proof (*cases isFault w'*)
      case *True*
      then obtain *f'* **where** *w'*: *w'=Fault f'*..
      with *Normal exec-c1*
      have *exec*: $\Gamma\vdash\langle$*Seq c1 c2*,*s*$\rangle$ =*n*$\Rightarrow$ *Fault f'*
        by (*auto intro*: *execn.intros*)
      from *w'-Fault-f w' noFault-w*
      have *f'* $\neq$ *f*
        by (*cases w*) *auto*
      moreover
      from *w' w'-Fault exec-mark-c2* **have** *isFault t*
        by (*auto dest*: *execn-Fault-end elim*: *isFaultE*)
      ultimately
      show *?thesis*
        using *exec*
        by *auto*
    next
      case *False*
      with *w'-noFault* **have** *w'*: *w'=w* **by** *simp*
      from *Seq.hyps exec-mark-c2*
      obtain *t'* **where**
        $\Gamma\vdash\langle$*c2*,*w*$\rangle$ =*n*$\Rightarrow$ *t'* **and**
        isFault t $\longrightarrow$ *isFault t'* **and**
        t' = *Fault f* $\longrightarrow$ *t'=t* **and**
        isFault t' $\longrightarrow$ *isFault t* **and**
        $\neg$ *isFault t'* $\longrightarrow$ *t'=t*
        by *blast*

```
      with Normal exec-c1 w'
      show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
qed
next
  case (Cond b c1 c2 s n t)
  have exec-mark: Γ⊢⟨mark-guards f (Cond b c1 c2),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-mark have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-mark have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec-mark have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
    show ?thesis
    proof (cases s'∈ b)
      case True
      with Normal exec-mark
      have Γ⊢⟨mark-guards f c1 ,Normal s'⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      with Normal True Cond.hyps obtain t'
        where Γ⊢⟨c1,Normal s'⟩ =n⇒ t'
            isFault t ⟶ isFault t'
            t' = Fault f ⟶ t'=t
            isFault t' ⟶ isFault t
            ¬ isFault t' ⟶ t' = t
        by blast
      with Normal True
      show ?thesis
        by (blast intro: execn.intros)
    next
      case False
      with Normal exec-mark
```

have $\Gamma\vdash\langle \textit{mark-guards } f\ c2\ ,\textit{Normal } s'\rangle =n\Rightarrow t$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Normal False Cond.hyps* **obtain** t'
where $\Gamma\vdash\langle c2,\textit{Normal } s'\rangle =n\Rightarrow t'$
$\textit{isFault } t \longrightarrow \textit{isFault } t'$
$t' = \textit{Fault } f \longrightarrow t'=t$
$\textit{isFault } t' \longrightarrow \textit{isFault } t$
$\neg\ \textit{isFault } t' \longrightarrow t' = t$
by *blast*
with *Normal False*
show *?thesis*
by (*blast intro*: *execn.intros*)
qed
qed
next
case (*While b c s n t*)
have *exec-mark*: $\Gamma\vdash\langle \textit{mark-guards } f\ (\textit{While } b\ c),s\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-mark* **have** $t=\textit{Fault } f$
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-mark* **have** $t=\textit{Stuck}$
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt* s')
with *exec-mark* **have** $t=\textit{Abrupt } s'$
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s')
{
fix $c'\ r\ w$
assume *exec-c'*: $\Gamma\vdash\langle c',r\rangle =n\Rightarrow w$
assume c': $c'=\textit{While } b\ (\textit{mark-guards } f\ c)$
have $\exists\, w'.\ \Gamma\vdash\langle \textit{While } b\ c,r\rangle =n\Rightarrow w' \wedge (\textit{isFault } w \longrightarrow \textit{isFault } w') \wedge$
$(w' = \textit{Fault } f \longrightarrow w'=w) \wedge (\textit{isFault } w' \longrightarrow \textit{isFault } w) \wedge$
$(\neg\ \textit{isFault } w' \longrightarrow w'=w)$
using *exec-c'* c'
proof (*induct*)
case (*WhileTrue* $r\ b'\ c''\ n\ u\ w$)
have *eqs*: $\textit{While } b'\ c'' = \textit{While } b\ (\textit{mark-guards } f\ c)$ **by** *fact*

from *WhileTrue.hyps eqs*
have *r-in-b*: $r \in b$ **by** *simp*
from *WhileTrue.hyps eqs*
have *exec-mark-c*: $\Gamma \vdash \langle mark\text{-}guards\ f\ c, Normal\ r\rangle =n\Rightarrow u$ **by** *simp*
from *WhileTrue.hyps eqs*
have *exec-mark-w*: $\Gamma \vdash \langle While\ b\ (mark\text{-}guards\ f\ c), u\rangle =n\Rightarrow w$
  by *simp*
show *?case*
proof –
  from *WhileTrue.hyps eqs* **have** $\Gamma \vdash \langle mark\text{-}guards\ f\ c, Normal\ r\rangle =n\Rightarrow u$
    by *simp*
  with *While.hyps*
  obtain u' **where**
    exec-c: $\Gamma \vdash \langle c, Normal\ r\rangle =n\Rightarrow u'$ **and**
    u-Fault: $isFault\ u \longrightarrow isFault\ u'$ **and**
    u′-Fault-f: $u' = Fault\ f \longrightarrow u' = u$ **and**
    u′-Fault: $isFault\ u' \longrightarrow isFault\ u$ **and**
    u′-noFault: $\neg\ isFault\ u' \longrightarrow u' = u$
    by *blast*
  show *?thesis*
  proof (*cases isFault* u')
    case *False*
    with *u′-noFault* **have** u': $u' = u$ **by** *simp*
    from *WhileTrue.hyps eqs* **obtain** w' **where**
      $\Gamma \vdash \langle While\ b\ c, u\rangle =n\Rightarrow w'$
      $isFault\ w \longrightarrow isFault\ w'$
      $w' = Fault\ f \longrightarrow w' = w$
      $isFault\ w' \longrightarrow isFault\ w$
      $\neg\ isFault\ w' \longrightarrow w' = w$
      by *blast*
    with u' *exec-c r-in-b*
    show *?thesis*
      by (*blast intro*: *execn.WhileTrue*)
  next
    case *True*
    then obtain f' **where** u': $u' = Fault\ f'$**..**
    with *exec-c r-in-b*
    have *exec*: $\Gamma \vdash \langle While\ b\ c, Normal\ r\rangle =n\Rightarrow Fault\ f'$
      by (*blast intro*: *execn.intros*)
    from *True u′-Fault* **have** *isFault u*
      by *simp*
    then obtain f **where** u: $u = Fault\ f$**..**
    with *exec-mark-w* **have** $w = Fault\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *exec* u' u *u′-Fault-f*
    show *?thesis*
      by *auto*
  qed
qed

```
  next
    case (WhileFalse r b′ c′′ n)
    have eqs: While b′  c′′ = While b (mark-guards f c) by fact
    from WhileFalse.hyps eqs
    have r-not-in-b: r∉b by simp
    show ?case
    proof −
      from r-not-in-b
      have Γ⊢⟨While b c,Normal r⟩ =n⇒ Normal r
        by (rule execn.WhileFalse)
      thus ?thesis
        by blast
    qed
  qed auto
} note hyp-while = this
show ?thesis
proof (cases s′∈b)
  case False
  with Normal exec-mark
  have t=s
    by (auto elim: execn-Normal-elim-cases)
  with Normal False show ?thesis
    by (auto intro: execn.intros)
next
  case True note s′-in-b = this
  with Normal exec-mark obtain r where
    exec-mark-c: Γ⊢⟨mark-guards f c,Normal s′⟩ =n⇒ r and
    exec-mark-w: Γ⊢⟨While b (mark-guards f c),r⟩ =n⇒ t
    by (auto elim: execn-Normal-elim-cases)
  from While.hyps exec-mark-c obtain r′ where
    exec-c: Γ⊢⟨c,Normal s′⟩ =n⇒ r′ and
    r-Fault: isFault r ⟶ isFault r′ and
    r′-Fault-f: r′ = Fault f ⟶ r′=r and
    r′-Fault: isFault r′ ⟶ isFault r and
    r′-noFault: ¬ isFault r′ ⟶ r′=r
    by blast
  show ?thesis
  proof (cases isFault r′)
    case False
    with r′-noFault have r′: r′=r by simp
    from hyp-while exec-mark-w
    obtain t′ where
      Γ⊢⟨While b c,r⟩ =n⇒ t′
      isFault t ⟶ isFault t′
      t′ = Fault f ⟶ t′=t
      isFault t′ ⟶ isFault t
      ¬ isFault t′ ⟶ t′=t
      by blast
    with r′ exec-c Normal s′-in-b
```

```
        show ?thesis
          by (blast intro: execn.intros)
      next
        case True
        then obtain f' where r': r'=Fault f'..
        hence Γ⊢⟨While b c,r'⟩ =n⇒ Fault f'
          by auto
        with Normal s'-in-b exec-c
        have exec: Γ⊢⟨While b c,Normal s'⟩ =n⇒ Fault f'
          by (auto intro: execn.intros)
        from True r'-Fault
        have isFault r
          by simp
        then obtain f where r: r=Fault f..
        with exec-mark-w have t=Fault f
          by (auto dest: execn-Fault-end)
        with Normal exec r' r r'-Fault-f
        show ?thesis
          by auto
      qed
    qed
  qed
next
  case Call thus ?case by auto
next
  case DynCom thus ?case
    by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
  case (Guard f' g c s n t)
  have exec-mark: Γ⊢⟨mark-guards f (Guard f' g c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-mark have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-mark have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec-mark have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
```

next
  case (*Normal s'*)
  show *?thesis*
  proof (*cases* $s' \in g$)
    case *False*
    with *Normal exec-mark* **have** *t*: $t = \mathit{Fault}\ f$
      by (*auto elim*: *execn-Normal-elim-cases*)
    from *False*
    have $\Gamma \vdash \langle \mathit{Guard}\ f'\ g\ c, \mathit{Normal}\ s' \rangle =n\Rightarrow \mathit{Fault}\ f'$
      by (*blast intro*: *execn.intros*)
    with *Normal t* **show** *?thesis*
      by *auto*
  next
    case *True*
    with *exec-mark Normal*
    have $\Gamma \vdash \langle \mathit{mark\text{-}guards}\ f\ c, \mathit{Normal}\ s' \rangle =n\Rightarrow t$
      by (*auto elim*: *execn-Normal-elim-cases*)
    with *Guard.hyps* **obtain** t' **where**
      $\Gamma \vdash \langle c, \mathit{Normal}\ s' \rangle =n\Rightarrow t'$ **and**
      $\mathit{isFault}\ t \longrightarrow \mathit{isFault}\ t'$ **and**
      $t' = \mathit{Fault}\ f \longrightarrow t' = t$ **and**
      $\mathit{isFault}\ t' \longrightarrow \mathit{isFault}\ t$ **and**
      $\neg\ \mathit{isFault}\ t' \longrightarrow t' = t$
      by *blast*
    with *Normal True*
    show *?thesis*
      by (*blast intro*: *execn.intros*)
  qed
qed
next
  case *Throw* **thus** *?case* **by** *auto*
next
  case (*Catch c1 c2 s n t*)
  have *exec-mark*: $\Gamma \vdash \langle \mathit{mark\text{-}guards}\ f\ (\mathit{Catch}\ c1\ c2), s \rangle =n\Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec-mark* **have** $t = \mathit{Fault}\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec-mark* **have** $t = \mathit{Stuck}$
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case (*Abrupt s'*)

with *exec-mark* **have** *t=Abrupt s′*
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s′*) **note** *s=this*
with *exec-mark* **have**
Γ⊢⟨*Catch* (*mark-guards f c1*) (*mark-guards f c2*),*Normal s′*⟩ =*n*⇒ *t* **by** *simp*
thus *?thesis*
proof (*cases*)
fix *w*
assume *exec-mark-c1*: Γ⊢⟨*mark-guards f c1*,*Normal s′*⟩ =*n*⇒ *Abrupt w*
assume *exec-mark-c2*: Γ⊢⟨*mark-guards f c2*,*Normal w*⟩ =*n*⇒ *t*
from *exec-mark-c1 Catch.hyps*
obtain *w′* **where**
exec-c1: Γ⊢⟨*c1*,*Normal s′*⟩ =*n*⇒ *w′* **and**
w′-Fault-f: *w′* = *Fault f* ⟶ *w′=Abrupt w* **and**
w′-Fault: *isFault w′* ⟶ *isFault* (*Abrupt w*) **and**
w′-noFault: ¬ *isFault w′* ⟶ *w′=Abrupt w*
by *fastforce*
show *?thesis*
proof (*cases w′*)
case (*Fault f′*)
with *Normal exec-c1* **have** Γ⊢⟨*Catch c1 c2*,*s*⟩ =*n*⇒ *Fault f′*
by (*auto intro*: *execn.intros*)
with *w′-Fault Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *w′-noFault* **have** *False*
by *simp*
thus *?thesis* **..**
next
case (*Normal w″*)
with *w′-noFault* **have** *False* **by** *simp* **thus** *?thesis* **..**
next
case (*Abrupt w″*)
with *w′-noFault* **have** *w″*: *w″=w* **by** *simp*
from *exec-mark-c2 Catch.hyps*
obtain *t′* **where**
Γ⊢⟨*c2*,*Normal w*⟩ =*n*⇒ *t′*
isFault t ⟶ *isFault t′*
t′ = *Fault f* ⟶ *t′=t*
isFault t′ ⟶ *isFault t*
¬ *isFault t′* ⟶ *t′=t*
by *blast*
with *w″ Abrupt s exec-c1*
show *?thesis*
by (*blast intro*: *execn.intros*)

qed
next
assume *t*: ¬ *isAbr t*
assume $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s'*$\rangle$ $=n\Rightarrow$ *t*
with *Catch.hyps*
obtain *t'* **where**
exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s'*$\rangle$ $=n\Rightarrow$ *t'* **and**
t-Fault: *isFault t* $\longrightarrow$ *isFault t'* **and**
t'-Fault-f: *t'* = *Fault f* $\longrightarrow$ *t'*=*t* **and**
t'-Fault: *isFault t'* $\longrightarrow$ *isFault t* **and**
t'-noFault: ¬ *isFault t'* $\longrightarrow$ *t'*=*t*
by *blast*
show *?thesis*
proof (*cases isFault t'*)
case *True*
then obtain *f'* **where** *t'*: *t'*=*Fault f'*..
with *exec-c1* **have** $\Gamma\vdash\langle$*Catch c1 c2*,*Normal s'*$\rangle$ $=n\Rightarrow$ *Fault f'*
by (*auto intro*: *execn.intros*)
with *t'-Fault-f t'-Fault t' s* **show** *?thesis*
by *auto*
next
case *False*
with *t'-noFault* **have** *t'*=*t* **by** *simp*
with *t exec-c1 s* **show** *?thesis*
by (*blast intro*: *execn.intros*)
qed
qed
qed
qed

lemma *exec-to-exec-mark-guards*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: ¬ *isFault t*
shows $\Gamma\vdash\langle$*mark-guards f c*,*s*$\rangle \Rightarrow t$
proof −
from *exec-to-execn* [*OF exec-c*] **obtain** *n* **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ ..
from *execn-to-execn-mark-guards* [*OF this t-not-Fault*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-to-exec-mark-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow$ *Fault f*
shows $\exists f'$. $\Gamma\vdash\langle$*mark-guards x c*,*s*$\rangle \Rightarrow$ *Fault f'*
proof −
from *exec-to-execn* [*OF exec-c*] **obtain** *n* **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow$ *Fault f* ..
from *execn-to-execn-mark-guards-Fault* [*OF this*]

show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-mark-guards-to-exec*:
assumes *exec-mark*: $\Gamma \vdash \langle mark\text{-}guards\ f\ c,s \rangle \Rightarrow t$
shows $\exists\, t'.\ \Gamma \vdash \langle c,s \rangle \Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge$
$(t' = Fault\ f \longrightarrow t'=t) \wedge$
$(isFault\ t' \longrightarrow isFault\ t) \wedge$
$(\neg\ isFault\ t' \longrightarrow t'=t)$
proof −
from *exec-to-execn* [*OF exec-mark*] **obtain** *n* **where**
$\Gamma \vdash \langle mark\text{-}guards\ f\ c,s \rangle =n\Rightarrow t$ **..**
from *execn-mark-guards-to-execn* [*OF this*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

2.7 Lemmas about *strip-guards*

lemma *execn-to-execn-strip-guards*:
assumes *exec-c*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma \vdash \langle strip\text{-}guards\ F\ c,s \rangle =n\Rightarrow t$
using *exec-c t-not-Fault* [*simplified not-isFault-iff*]
by (*induct*) (*auto intro*: *execn.intros dest*: *noFaultn-startD′*)

lemma *execn-to-execn-strip-guards-Fault*:
assumes *exec-c*: $\Gamma \vdash \langle c,s \rangle =n\Rightarrow t$
shows $\bigwedge f.\ [\![t=Fault\ f;\ f \notin F]\!] \Longrightarrow \Gamma \vdash \langle strip\text{-}guards\ F\ c,s \rangle =n\Rightarrow Fault\ f$
using *exec-c*
proof (*induct*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Guard* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case *SpecStuck* **thus** *?case* **by** *auto*
next

```
  case (Seq c1 s n w c2 t)
  have exec-c1: Γ⊢⟨c1,Normal s⟩ =n⇒ w by fact
  have exec-c2: Γ⊢⟨c2,w⟩ =n⇒ t by fact
  have t: t=Fault f by fact
  have notinF: f ∉ F by fact
  show ?case
  proof (cases w)
    case (Fault f')
    with exec-c2 t have f'=f
      by (auto dest: execn-Fault-end)
    with Fault notinF Seq.hyps
    have Γ⊢⟨strip-guards F c1,Normal s⟩ =n⇒ Fault f
      by auto
    moreover have Γ⊢⟨strip-guards F c2,Fault f⟩ =n⇒ Fault f
      by auto
    ultimately show ?thesis
      by (auto intro: execn.intros)
  next
    case (Normal s')
    with execn-to-execn-strip-guards [OF exec-c1]
    have exec-strip-c1: Γ⊢⟨strip-guards F c1,Normal s⟩ =n⇒ w
      by simp
    with Seq.hyps t notinF
    have Γ⊢⟨strip-guards F c2,w⟩ =n⇒ Fault f
      by blast
    with exec-strip-c1 show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-strip-guards [OF exec-c1]
    have exec-strip-c1: Γ⊢⟨strip-guards F c1,Normal s⟩ =n⇒ w
      by simp
    with Seq.hyps t notinF
    have Γ⊢⟨strip-guards F c2,w⟩ =n⇒ Fault f
      by (auto intro: execn.intros)
    with exec-strip-c1 show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-c2 have t=Stuck
      by (auto dest: execn-Stuck-end)
    with t show ?thesis by simp
  qed
next
  case CondTrue thus ?case by (fastforce intro: execn.intros)
next
  case CondFalse thus ?case by (fastforce intro: execn.intros)
next
  case (WhileTrue s b c n w t)
```

```
  have exec-c: Γ⊢⟨c,Normal s⟩ =n⇒ w by fact
  have exec-w: Γ⊢⟨While b c,w⟩ =n⇒ t by fact
  have t: t = Fault f by fact
  have notinF: f ∉ F by fact
  have s-in-b: s ∈ b by fact
  show ?case
  proof (cases w)
    case (Fault f')
    with exec-w t have f'=f
      by (auto dest: execn-Fault-end)
    with Fault notinF WhileTrue.hyps
    have Γ⊢⟨strip-guards F c,Normal s⟩ =n⇒ Fault f
      by auto
    moreover have Γ⊢⟨strip-guards F (While b c),Fault f⟩ =n⇒ Fault f
      by auto
    ultimately show ?thesis
      using s-in-b by (auto intro: execn.intros)
  next
    case (Normal s')
    with execn-to-execn-strip-guards [OF exec-c]
    have exec-strip-c: Γ⊢⟨strip-guards F c,Normal s⟩ =n⇒ w
      by simp
    with WhileTrue.hyps t notinF
    have Γ⊢⟨strip-guards F (While b c),w⟩ =n⇒ Fault f
      by blast
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-strip-guards [OF exec-c]
    have exec-strip-c: Γ⊢⟨strip-guards F c,Normal s⟩ =n⇒ w
      by simp
    with WhileTrue.hyps t notinF
    have Γ⊢⟨strip-guards F (While b c),w⟩ =n⇒ Fault f
      by (auto intro: execn.intros)
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-w have t=Stuck
      by (auto dest: execn-Stuck-end)
    with t show ?thesis by simp
  qed
next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
next
  case Call thus ?case by (fastforce intro: execn.intros)
next
  case CallUndefined thus ?case by simp
```

next
  case *StuckProp* **thus** *?case* **by** *simp*
next
  case *DynCom* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *Throw* **thus** *?case* **by** *simp*
next
  case *AbruptProp* **thus** *?case* **by** *simp*
next
  case (*CatchMatch c1 s n w c2 t*)
  have *exec-c1*: $\Gamma\vdash\langle c1, Normal\ s\rangle =n\Rightarrow Abrupt\ w$ **by** *fact*
  have *exec-c2*: $\Gamma\vdash\langle c2, Normal\ w\rangle =n\Rightarrow t$ **by** *fact*
  have *t*: $t = Fault\ f$ **by** *fact*
  have *notinF*: $f \notin F$ **by** *fact*
  from *execn-to-execn-strip-guards* [*OF exec-c1*]
  have *exec-strip-c1*: $\Gamma\vdash\langle strip\text{-}guards\ F\ c1, Normal\ s\rangle =n\Rightarrow Abrupt\ w$
    by *simp*
  with *CatchMatch.hyps t notinF*
  have $\Gamma\vdash\langle strip\text{-}guards\ F\ c2, Normal\ w\rangle =n\Rightarrow Fault\ f$
    by *blast*
  with *exec-strip-c1* **show** *?case*
    by (*auto intro*: *execn.intros*)
next
  case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
qed

lemma *execn-to-execn-strip-guards'*:
 assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
 assumes *t-not-Fault*: $t \notin Fault\ `\ F$
 shows $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow t$
proof (*cases t*)
  case (*Fault f*)
  with *t-not-Fault exec-c* **show** *?thesis*
    by (*auto intro*: *execn-to-execn-strip-guards-Fault*)
qed (*insert exec-c*, *auto intro*: *execn-to-execn-strip-guards*)

lemma *execn-strip-guards-to-execn*:
  $\bigwedge s\ n\ t.\ \Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow t$
  $\Longrightarrow \exists t'.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t' \land$
           $(isFault\ t \longrightarrow isFault\ t') \land$
           $(t' \in Fault\ `\ (-\ F) \longrightarrow t'=t) \land$
           $(\neg\ isFault\ t' \longrightarrow t'=t)$
proof (*induct c*)
  case *Skip* **thus** *?case* **by** *auto*
next
  case *Basic* **thus** *?case* **by** *auto*
next
  case *Spec* **thus** *?case* **by** *auto*
next

```
case (Seq c1 c2 s n t)
have exec-strip: Γ⊢⟨strip-guards F (Seq c1 c2),s⟩ =n⇒ t by fact
then obtain w where
  exec-strip-c1: Γ⊢⟨strip-guards F c1,s⟩ =n⇒ w and
  exec-strip-c2: Γ⊢⟨strip-guards F c2,w⟩ =n⇒ t
  by (auto elim: execn-elim-cases)
from Seq.hyps exec-strip-c1
obtain w' where
  exec-c1: Γ⊢⟨c1,s⟩ =n⇒ w' and
  w-Fault: isFault w ⟶ isFault w' and
  w'-Fault: w' ∈ Fault ` (− F) ⟶ w'=w and
  w'-noFault: ¬ isFault w' ⟶ w'=w
  by blast
show ?case
proof (cases s)
  case (Fault f)
  with exec-strip have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
  case Stuck
  with exec-strip have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec-strip have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  show ?thesis
  proof (cases isFault w)
    case True
    then obtain f where w': w=Fault f..
    moreover with exec-strip-c2
    have t: t=Fault f
      by (auto dest: execn-Fault-end)
    ultimately show ?thesis
      using Normal w-Fault w'-Fault exec-c1
      by (fastforce intro: execn.intros elim: isFaultE)
  next
    case False
    note noFault-w = this
    show ?thesis
    proof (cases isFault w')
```

```
      case True
      then obtain f′ where w′: w′=Fault f′..
      with Normal exec-c1
      have exec: Γ⊢⟨Seq c1 c2,s⟩ =n⇒ Fault f′
        by (auto intro: execn.intros)
      from w′-Fault w′ noFault-w
      have f′ ∈ F
        by (cases w) auto
      with exec
      show ?thesis
        by auto
    next
      case False
      with w′-noFault have w′: w′=w by simp
      from Seq.hyps exec-strip-c2
      obtain t′ where
        Γ⊢⟨c2,w⟩ =n⇒ t′ and
        isFault t ⟶ isFault t′ and
        t′ ∈ Fault ‘ (−F) ⟶ t′=t and
        ¬ isFault t′ ⟶ t′=t
        by blast
      with Normal exec-c1 w′
      show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
 qed
next
next
 case (Cond b c1 c2 s n t)
 have exec-strip: Γ⊢⟨strip-guards F (Cond b c1 c2),s⟩ =n⇒ t by fact
 show ?case
 proof (cases s)
  case (Fault f)
  with exec-strip have t=Fault f
   by (auto dest: execn-Fault-end)
  with Fault show ?thesis
   by auto
 next
  case Stuck
  with exec-strip have t=Stuck
   by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
   by auto
 next
  case (Abrupt s′)
  with exec-strip have t=Abrupt s′
   by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
```

```
      by auto
  next
    case (Normal s')
    show ?thesis
    proof (cases s'∈ b)
      case True
      with Normal exec-strip
      have Γ⊢⟨strip-guards F c1 ,Normal s'⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      with Normal True Cond.hyps obtain t'
        where Γ⊢⟨c1,Normal s'⟩ =n⇒ t'
            isFault t ⟶ isFault t'
            t' ∈ Fault ‘ (−F) ⟶ t'=t
            ¬ isFault t' ⟶ t' = t
        by blast
      with Normal True
      show ?thesis
        by (blast intro: execn.intros)
    next
      case False
      with Normal exec-strip
      have Γ⊢⟨strip-guards F c2 ,Normal s'⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      with Normal False Cond.hyps obtain t'
        where Γ⊢⟨c2,Normal s'⟩ =n⇒ t'
            isFault t  ⟶ isFault t'
            t' ∈ Fault ‘ (−F) ⟶ t'=t
            ¬ isFault t' ⟶ t' = t
        by blast
      with Normal False
      show ?thesis
        by (blast intro: execn.intros)
    qed
  qed
next
  case (While b c s n t)
  have exec-strip: Γ⊢⟨strip-guards F (While b c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-strip have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-strip have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
```

```
    by auto
next
  case (Abrupt s')
  with exec-strip have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  {
    fix c' r w
    assume exec-c': Γ⊢⟨c',r⟩ =n⇒ w
    assume c': c'=While b (strip-guards F c)
    have ∃ w'. Γ⊢⟨While b c,r⟩ =n⇒ w' ∧ (isFault w ⟶ isFault w') ∧
               (w' ∈ Fault ` (−F) ⟶ w'=w) ∧
               (¬ isFault w' ⟶ w'=w)
      using exec-c' c'
    proof (induct)
      case (WhileTrue r b' c'' n u w)
      have eqs: While b' c'' = While b (strip-guards F c) by fact
      from WhileTrue.hyps eqs
      have r-in-b: r∈b by simp
      from WhileTrue.hyps eqs
      have exec-strip-c: Γ⊢⟨strip-guards F c,Normal r⟩ =n⇒ u by simp
      from WhileTrue.hyps eqs
      have exec-strip-w: Γ⊢⟨While b (strip-guards F c),u⟩ =n⇒ w
        by simp
      show ?case
      proof −
        from WhileTrue.hyps eqs have Γ⊢⟨strip-guards F c,Normal r⟩ =n⇒ u
          by simp
        with While.hyps
        obtain u' where
          exec-c: Γ⊢⟨c,Normal r⟩ =n⇒ u' and
          u-Fault: isFault u ⟶ isFault u' and
          u'-Fault: u' ∈ Fault ` (−F) ⟶ u'=u and
          u'-noFault: ¬ isFault u' ⟶ u'=u
          by blast
        show ?thesis
        proof (cases isFault u')
          case False
          with u'-noFault have u': u'=u by simp
          from WhileTrue.hyps eqs obtain w' where
            Γ⊢⟨While b c,u⟩ =n⇒ w'
            isFault w  ⟶ isFault w'
            w' ∈ Fault ` (−F) ⟶ w'=w
            ¬ isFault w' ⟶ w' = w
            by blast
          with u' exec-c r-in-b
```

```
        show ?thesis
          by (blast intro: execn.WhileTrue)
      next
        case True
        then obtain f' where u': u'=Fault f'..
        with exec-c r-in-b
        have exec: Γ⊢⟨While b c,Normal r⟩ =n⇒ Fault f'
          by (blast intro: execn.intros)
        show ?thesis
        proof (cases isFault u)
          case True
          then obtain f where u: u=Fault f..
          with exec-strip-w have w=Fault f
            by (auto dest: execn-Fault-end)
          with exec u' u u'-Fault
          show ?thesis
            by auto
        next
          case False
          with u'-Fault u' have f' ∈ F
            by (cases u) auto
          with exec show ?thesis
            by auto
        qed
      qed
    qed
  next
    case (WhileFalse r b' c'' n)
    have eqs: While b'  c'' = While b (strip-guards F c) by fact
    from WhileFalse.hyps eqs
    have r-not-in-b: r∉b by simp
    show ?case
    proof −
      from r-not-in-b
      have Γ⊢⟨While b c,Normal r⟩ =n⇒ Normal r
        by (rule execn.WhileFalse)
      thus ?thesis
        by blast
    qed
  qed auto
} note hyp-while = this
show ?thesis
proof (cases s'∈b)
  case False
  with Normal exec-strip
  have t=s
    by (auto elim: execn-Normal-elim-cases)
  with Normal False show ?thesis
    by (auto intro: execn.intros)
```

next
  case *True* **note** s'*-in-b* = *this*
  with *Normal exec-strip* **obtain** *r* **where**
    exec-strip-c: $\Gamma\vdash\langle$*strip-guards F c*,*Normal* $s'\rangle$ $=n\Rightarrow$ *r* **and**
    exec-strip-w: $\Gamma\vdash\langle$*While b* (*strip-guards F c*),$r\rangle$ $=n\Rightarrow$ *t*
    by (*auto elim*: *execn-Normal-elim-cases*)
  from *While.hyps exec-strip-c* **obtain** r' **where**
    exec-c: $\Gamma\vdash\langle c$,*Normal* $s'\rangle$ $=n\Rightarrow$ r' **and**
    r-Fault: *isFault r* $\longrightarrow$ *isFault* r' **and**
    r'*-Fault*: $r' \in$ *Fault* ‘ $(-F)$ $\longrightarrow$ $r'=r$ **and**
    r'*-noFault*: $\neg$ *isFault* r' $\longrightarrow$ $r'=r$
    by *blast*
  show *?thesis*
  proof (*cases isFault* r')
    case *False*
    with r'*-noFault* **have** r': $r'=r$ **by** *simp*
    from *hyp-while exec-strip-w*
    obtain t' **where**
      $\Gamma\vdash\langle$*While b c*,$r\rangle$ $=n\Rightarrow$ t'
      isFault t $\longrightarrow$ *isFault* t'
      $t' \in$ *Fault* ‘ $(-F)$ $\longrightarrow$ $t'=t$
      $\neg$ *isFault* t' $\longrightarrow$ $t'=t$
      by *blast*
    with r' *exec-c Normal* s'*-in-b*
    show *?thesis*
      by (*blast intro*: *execn.intros*)
  next
    case *True*
    then obtain f' **where** r': r'=*Fault* f'**..**
    hence $\Gamma\vdash\langle$*While b c*,$r'\rangle$ $=n\Rightarrow$ *Fault* f'
      by *auto*
    with *Normal* s'*-in-b exec-c*
    have *exec*: $\Gamma\vdash\langle$*While b c*,*Normal* $s'\rangle$ $=n\Rightarrow$ *Fault* f'
      by (*auto intro*: *execn.intros*)
    show *?thesis*
    proof (*cases isFault r*)
      case *True*
      then obtain *f* **where** *r*: *r*=*Fault f***..**
      with *exec-strip-w* **have** *t*=*Fault f*
        by (*auto dest*: *execn-Fault-end*)
      with *Normal exec* r' *r* r'*-Fault*
      show *?thesis*
        by *auto*
    next
      case *False*
      with r'*-Fault* r' **have** $f' \in F$
        by (*cases r*) *auto*
      with *Normal exec* **show** *?thesis*
        by *auto*

```
        qed
      qed
    qed
  qed
next
  case Call thus ?case by auto
next
  case DynCom thus ?case
    by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
  case (Guard f g c s n t)
  have exec-strip: Γ⊢⟨strip-guards F (Guard f g c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-strip have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-strip have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec-strip have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
    show ?thesis
    proof (cases f∈F)
      case True
      with exec-strip Normal
      have exec-strip-c: Γ⊢⟨strip-guards F c,Normal s'⟩ =n⇒ t
        by simp
      with Guard.hyps obtain t' where
        Γ⊢⟨c,Normal s'⟩ =n⇒ t' and
        isFault t ⟶ isFault t' and
        t' ∈ Fault ` (−F) ⟶ t'=t and
        ¬ isFault t' ⟶ t'=t
        by blast
      with Normal True
      show ?thesis
        by (cases s'∈ g) (fastforce intro: execn.intros)+
    next
```

```
      case False
      note f-notin-F = this
      show ?thesis
      proof (cases s'∈g)
        case False
        with Normal exec-strip f-notin-F have t: t=Fault f
          by (auto elim: execn-Normal-elim-cases)
        from False
        have Γ⊢⟨Guard f g c,Normal s'⟩ =n⇒ Fault f
          by (blast intro: execn.intros)
        with False Normal t show ?thesis
          by auto
      next
        case True
        with exec-strip Normal f-notin-F
        have Γ⊢⟨strip-guards F c,Normal s'⟩ =n⇒ t
          by (auto elim: execn-Normal-elim-cases)
        with Guard.hyps obtain t' where
          Γ⊢⟨c,Normal s'⟩ =n⇒ t' and
          isFault t ⟶ isFault t' and
          t' ∈ Fault ` (−F) ⟶ t'=t and
          ¬ isFault t' ⟶ t'=t
          by blast
        with Normal True
        show ?thesis
          by (blast intro: execn.intros)
      qed
    qed
  qed
next
  case Throw thus ?case by auto
next
  case (Catch c1 c2 s n t)
  have exec-strip: Γ⊢⟨strip-guards F (Catch c1 c2),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-strip have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-strip have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
```

with *exec-strip* **have** t=*Abrupt* s'
  by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
  by *auto*
next
case (*Normal* s') **note** s=*this*
with *exec-strip* **have**
$\Gamma\vdash\langle$*Catch* (*strip-guards F c1*) (*strip-guards F c2*),*Normal* $s'\rangle$ $=n\Rightarrow$ t **by** *simp*
thus *?thesis*
proof (*cases*)
  fix w
  assume *exec-strip-c1*: $\Gamma\vdash\langle$*strip-guards F c1*,*Normal* $s'\rangle$ $=n\Rightarrow$ *Abrupt* w
  assume *exec-strip-c2*: $\Gamma\vdash\langle$*strip-guards F c2*,*Normal* $w\rangle$ $=n\Rightarrow$ t
  from *exec-strip-c1 Catch.hyps*
  obtain w' **where**
    exec-c1: $\Gamma\vdash\langle$*c1*,*Normal* $s'\rangle$ $=n\Rightarrow$ w' **and**
    w'*-Fault*: $w' \in$ *Fault* ` $(-F)$ $\longrightarrow$ w'=*Abrupt* w **and**
    w'*-noFault*: $\neg$ *isFault* w' $\longrightarrow$ w'=*Abrupt* w
    by *blast*
  show *?thesis*
  proof (*cases* w')
    case (*Fault* f')
    with *Normal exec-c1* **have** $\Gamma\vdash\langle$*Catch c1 c2*,$s\rangle$ $=n\Rightarrow$ *Fault* f'
      by (*auto intro*: *execn.intros*)
    with w'*-Fault Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with w'*-noFault* **have** *False*
      by *simp*
    thus *?thesis* **..**
  next
    case (*Normal* w'')
    with w'*-noFault* **have** *False* **by** *simp* **thus** *?thesis* **..**
  next
    case (*Abrupt* w'')
    with w'*-noFault* **have** w'': w''=w **by** *simp*
    from *exec-strip-c2 Catch.hyps*
    obtain t' **where**
      $\Gamma\vdash\langle$*c2*,*Normal* $w\rangle$ $=n\Rightarrow$ t'
      isFault t $\longrightarrow$ *isFault* t'
      $t' \in$ *Fault* ` $(-F)$ $\longrightarrow$ t'=t
      $\neg$ *isFault* t' $\longrightarrow$ t'=t
      by *blast*
    with w'' *Abrupt s exec-c1*
    show *?thesis*
      by (*blast intro*: *execn.intros*)
  qed
next

assume t: $\neg$ *isAbr t*
assume $\Gamma\vdash\langle$*strip-guards F c1*,*Normal s'*$\rangle$ $=n\Rightarrow$ t
with *Catch.hyps*
obtain t' **where**
exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s'*$\rangle$ $=n\Rightarrow$ t' **and**
t-Fault: *isFault t* $\longrightarrow$ *isFault t'* **and**
t'-Fault: $t' \in$ *Fault* ‘ $(-F)$ $\longrightarrow$ $t'=t$ **and**
t'-noFault: $\neg$ *isFault t'* $\longrightarrow$ $t'=t$
by *blast*
show *?thesis*
proof (*cases isFault t'*)
case *True*
then obtain f' **where** t': $t'=$*Fault f'*..
with *exec-c1* **have** $\Gamma\vdash\langle$*Catch c1 c2*,*Normal s'*$\rangle$ $=n\Rightarrow$ *Fault f'*
by (*auto intro*: *execn.intros*)
with *t'-Fault t' s* **show** *?thesis*
by *auto*
next
case *False*
with *t'-noFault* **have** $t'=t$ **by** *simp*
with *t exec-c1 s* **show** *?thesis*
by (*blast intro*: *execn.intros*)
qed
qed
qed
qed

lemma *execn-strip-to-execn*:
assumes *exec-strip*: *strip F* $\Gamma\vdash\langle c,s\rangle$ $=n\Rightarrow$ t
shows $\exists\, t'.\ \Gamma\vdash\langle c,s\rangle$ $=n\Rightarrow$ t' $\wedge$
(*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$
($t' \in$ *Fault* ‘ $(-\ F)$ $\longrightarrow$ $t'=t$) $\wedge$
($\neg$ *isFault t'* $\longrightarrow$ $t'=t$)
using *exec-strip*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *Guard* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *Basic* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *Spec* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
case *SpecStuck* **thus** *?case* **by** (*blast intro*: *execn.intros*)

```
next
  case Seq thus ?case by (blast intro: execn.intros elim: isFaultE)
next
  case CondTrue thus ?case by (blast intro: execn.intros)
next
  case CondFalse thus ?case by (blast intro: execn.intros)
next
  case WhileTrue thus ?case by (blast intro: execn.intros elim: isFaultE)
next
  case WhileFalse thus ?case by (blast intro: execn.intros)
next
  case Call thus ?case
    by simp (blast intro: execn.intros dest: execn-strip-guards-to-execn)
next
  case CallUndefined thus ?case
    by simp (blast intro: execn.intros)
next
  case StuckProp thus ?case
    by blast
next
  case DynCom thus ?case by (blast intro: execn.intros)
next
  case Throw thus ?case by (blast intro: execn.intros)
next
  case AbruptProp thus ?case by (blast intro: execn.intros)
next
  case (CatchMatch c1 s n r c2 t)
  then obtain r' t' where
    exec-c1: Γ⊢⟨c1,Normal s⟩ =n⇒ r'  and
    r'-Fault: r' ∈ Fault ` (−F) ⟶ r' = Abrupt r and
    r'-noFault: ¬ isFault r' ⟶ r' = Abrupt r and
    exec-c2: Γ⊢⟨c2,Normal r⟩ =n⇒ t' and
    t-Fault: isFault t ⟶ isFault t' and
    t'-Fault: t' ∈ Fault ` (−F) ⟶ t' = t and
    t'-noFault: ¬ isFault t' ⟶ t' = t
    by blast
  show ?case
  proof (cases isFault r')
    case True
    then obtain f' where r': r'=Fault f'..
    with exec-c1 have Γ⊢⟨Catch c1 c2,Normal s⟩ =n⇒ Fault f'
      by (auto intro: execn.intros)
    with r' r'-Fault show ?thesis
      by (auto intro: execn.intros)
  next
    case False
    with r'-noFault have r'=Abrupt r by simp
    with exec-c1 exec-c2 t-Fault t'-noFault t'-Fault
    show ?thesis
```

by (*blast intro*: *execn.intros*)
qed
next
case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros elim*: *isFaultE*)
qed

lemma *exec-strip-guards-to-exec*:
assumes *exec-strip*: $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
shows $\exists\, t'.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge$
$(t' \in Fault\ `\ (-F) \longrightarrow t'=t) \wedge$
$(\neg\ isFault\ t' \longrightarrow t'=t)$
proof −
from *exec-strip* **obtain** *n* **where**
execn-strip: $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ $=n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
then obtain t' **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow t'$
$isFault\ t \longrightarrow isFault\ t'\ t' \in Fault\ `\ (-F) \longrightarrow t'=t\ \neg\ isFault\ t' \longrightarrow t'=t$
by (*blast dest*: *execn-strip-guards-to-execn*)
thus *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-strip-to-exec*:
assumes *exec-strip*: *strip F* $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
shows $\exists\, t'.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge$
$(t' \in Fault\ `\ (-F) \longrightarrow t'=t) \wedge$
$(\neg\ isFault\ t' \longrightarrow t'=t)$
proof −
from *exec-strip* **obtain** *n* **where**
execn-strip: *strip F* $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
then obtain t' **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow t'$
$isFault\ t \longrightarrow isFault\ t'\ t' \in Fault\ `\ (-F) \longrightarrow t'=t\ \neg\ isFault\ t' \longrightarrow t'=t$
by (*blast dest*: *execn-strip-to-execn*)
thus *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$

by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
by (*rule execn-to-execn-strip-guards*)
thus $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ $\Rightarrow$ *t*
by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards'*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
shows $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ $\Rightarrow$ *t*
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$*t*
by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
by (*rule execn-to-execn-strip-guards'*)
thus $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle$ $\Rightarrow$ *t*
by (*rule execn-to-exec*)
qed

lemma *execn-to-execn-strip*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$ *t*
assumes *t-not-Fault*: ¬ *isFault t*
shows *strip F* $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$ *t*
using *exec-c t-not-Fault*
proof (*induct*)
case (*Call p bdy s n s'*)
have *bdy*: Γ *p* = *Some bdy* **by** *fact*
from *Call* **have** *strip F* $\Gamma\vdash\langle$*bdy*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *s'*
by *blast*
from *execn-to-execn-strip-guards* [*OF this*] *Call*
have *strip F* $\Gamma\vdash\langle$*strip-guards F bdy*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *s'*
by *simp*
moreover from *bdy* **have** (*strip F* Γ) *p* = *Some* (*strip-guards F bdy*)
by *simp*
ultimately
show *?case*
by (*blast intro*: *execn.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*auto intro*: *execn.CallUndefined*)
qed (*auto intro*: *execn.intros dest*: *noFaultn-startD' simp add*: *not-isFault-iff*)

lemma *execn-to-execn-strip'*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$ *t*
assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
shows *strip F* $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$ *t*
using *exec-c t-not-Fault*

proof (*induct*)
  case (*Call p bdy s n s'*)
  have *bdy*: Γ *p* = *Some bdy* **by** *fact*
  from *Call* **have** *strip F* Γ⊢⟨*bdy*,*Normal s*⟩ =*n*⇒ *s'*
    by *blast*
  from *execn-to-execn-strip-guards'* [*OF this*] *Call*
  have *strip F* Γ⊢⟨*strip-guards F bdy*,*Normal s*⟩ =*n*⇒ *s'*
    by *simp*
  moreover from *bdy* **have** (*strip F* Γ) *p* = *Some* (*strip-guards F bdy*)
    by *simp*
  ultimately
  show *?case*
    by (*blast intro*: *execn.intros*)
next
  case *CallUndefined* **thus** *?case* **by** (*auto intro*: *execn.CallUndefined*)
next
  case (*Seq c1 s n s' c2 t*)
  show *?case*
  proof (*cases isFault s'*)
    case *False*
    with *Seq* **show** *?thesis*
      by (*auto intro*: *execn.intros simp add*: *not-isFault-iff*)
  next
    case *True*
    then obtain *f'* **where** *s'*: *s'*=*Fault f'* **by** (*auto simp add*: *isFault-def*)
    with *Seq* **obtain** *t*=*Fault f'* **and** *f'* ∉ *F*
      by (*force dest*: *execn-Fault-end*)
    with *Seq s'* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  qed
next
  case (*WhileTrue b c s n s' t*)
  show *?case*
  proof (*cases isFault s'*)
    case *False*
    with *WhileTrue* **show** *?thesis*
      by (*auto intro*: *execn.intros simp add*: *not-isFault-iff*)
  next
    case *True*
    then obtain *f'* **where** *s'*: *s'*=*Fault f'* **by** (*auto simp add*: *isFault-def*)
    with *WhileTrue* **obtain** *t*=*Fault f'* **and** *f'* ∉ *F*
      by (*force dest*: *execn-Fault-end*)
    with *WhileTrue s'* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  qed
qed (*auto intro*: *execn.intros*)

lemma *exec-to-exec-strip*:
 assumes *exec-c*: Γ⊢⟨*c*,*s*⟩ ⇒ *t*

assumes *t-not-Fault*: ¬ *isFault t*
shows *strip F* $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
proof −
  from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
    by (*auto simp add*: *exec-iff-execn*)
  from *this t-not-Fault*
  have *strip F* $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
    by (*rule execn-to-execn-strip*)
  thus *strip F* $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
    by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip′*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
shows *strip F* $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
proof −
  from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
    by (*auto simp add*: *exec-iff-execn*)
  from *this t-not-Fault*
  have *strip F* $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
    by (*rule execn-to-execn-strip′*)
  thus *strip F* $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
    by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow$ *Fault f*
assumes *f-notin-F*: $f \notin F$
shows$\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle \Rightarrow$ *Fault f*
proof −
  from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow$*Fault f*
    by (*auto simp add*: *exec-iff-execn*)
  from *execn-to-execn-strip-guards-Fault* [*OF this - f-notin-F*]
  have $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle =n\Rightarrow$ *Fault f*
    by *simp*
  thus $\Gamma\vdash\langle$*strip-guards F c*,*s*$\rangle \Rightarrow$ *Fault f*
    by (*rule execn-to-exec*)
qed

2.8 Lemmas about $c_1 \cap_g c_2$

lemma *inter-guards-execn-Normal-noFault*:
  $\bigwedge$*c c2 s t n*. ⟦(*c1* $\cap_g$ *c2*) = *Some c*; $\Gamma\vdash\langle c,$*Normal s*$\rangle =n\Rightarrow t$; ¬ *isFault t*⟧
    $\Longrightarrow \Gamma\vdash\langle$*c1*,*Normal s*$\rangle =n\Rightarrow t \wedge \Gamma\vdash\langle$*c2*,*Normal s*$\rangle =n\Rightarrow t$
proof (*induct c1*)
  case *Skip*
  have (*Skip* $\cap_g$ *c2*) = *Some c* **by** *fact*
  then obtain *c2*: *c2=Skip* **and** *c*: *c=Skip*

```
    by (simp add: inter-guards-Skip)
  have Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  with c have t=Normal s
    by (auto elim: execn-Normal-elim-cases)
  with Skip c2
  show ?case
    by (auto intro: execn.intros)
next
  case (Basic f)
  have (Basic f ∩g c2) = Some c by fact
  then obtain c2: c2=Basic f and c: c=Basic f
    by (simp add: inter-guards-Basic)
  have Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  with c have t=Normal (f s)
    by (auto elim: execn-Normal-elim-cases)
  with Basic c2
  show ?case
    by (auto intro: execn.intros)
next
  case (Spec r)
  have (Spec r ∩g c2) = Some c by fact
  then obtain c2: c2=Spec r and c: c=Spec r
    by (simp add: inter-guards-Spec)
  have Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  with c have Γ⊢⟨Spec r,Normal s⟩ =n⇒ t by simp
  from this Spec c2 show ?case
    by (cases) (auto intro: execn.intros)
next
  case (Seq a1 a2)
  have noFault: ¬ isFault t by fact
  have (Seq a1 a2 ∩g c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Seq b1 b2 and
    d1: (a1 ∩g b1) = Some d1 and d2: (a2 ∩g b2) = Some d2 and
    c: c=Seq d1 d2
    by (auto simp add: inter-guards-Seq)
  have Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  with c obtain s' where
    exec-d1: Γ⊢⟨d1,Normal s⟩ =n⇒ s' and
    exec-d2: Γ⊢⟨d2,s'⟩ =n⇒ t
    by (auto elim: execn-Normal-elim-cases)
  show ?case
  proof (cases s')
    case (Fault f')
    with exec-d2 have t=Fault f'
      by (auto intro: execn-Fault-end)
    with noFault show ?thesis by simp
  next
    case (Normal s'')
```

with *d1 exec-d1 Seq.hyps*
obtain
$\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Normal\ s''$ **and** $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Normal\ s''$
by *auto*
moreover
from *Normal d2 exec-d2 noFault Seq.hyps*
obtain $\Gamma\vdash\langle a2,Normal\ s''\rangle =n\Rightarrow t$ **and** $\Gamma\vdash\langle b2,Normal\ s''\rangle =n\Rightarrow t$
by *auto*
ultimately
show *?thesis*
using *Normal c2* **by** (*auto intro*: *execn.intros*)
next
case (*Abrupt s''*)
with *exec-d2* **have** *t=Abrupt s''*
by (*auto simp add*: *execn-Abrupt-end*)
moreover
from *Abrupt d1 exec-d1 Seq.hyps*
obtain $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Abrupt\ s''$ **and** $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Abrupt$
s''
by *auto*
moreover
obtain
$\Gamma\vdash\langle a2,Abrupt\ s''\rangle =n\Rightarrow Abrupt\ s''$ **and** $\Gamma\vdash\langle b2,Abrupt\ s''\rangle =n\Rightarrow Abrupt\ s''$
by *auto*
ultimately
show *?thesis*
using *Abrupt c2* **by** (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-d2* **have** *t=Stuck*
by (*auto simp add*: *execn-Stuck-end*)
moreover
from *Stuck d1 exec-d1 Seq.hyps*
obtain $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Stuck$ **and** $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Stuck$
by *auto*
moreover
obtain
$\Gamma\vdash\langle a2,Stuck\rangle =n\Rightarrow Stuck$ **and** $\Gamma\vdash\langle b2,Stuck\rangle =n\Rightarrow Stuck$
by *auto*
ultimately
show *?thesis*
using *Stuck c2* **by** (*auto intro*: *execn.intros*)
qed
next
case (*Cond b t1 e1*)
have *noFault*: $\neg$ *isFault t* **by** *fact*
have (*Cond b t1 e1* $\cap_g$ *c2*) = *Some c* **by** *fact*
then obtain *t2 e2 t3 e3* **where**
c2: *c2=Cond b t2 e2* **and**

```
    t3: (t1 ∩g t2) = Some t3 and
    e3: (e1 ∩g e2) = Some e3 and
    c: c=Cond b t3 e3
    by (auto simp add: inter-guards-Cond)
  have Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  with c have Γ⊢⟨Cond b t3 e3,Normal s⟩ =n⇒ t
    by simp
  then show ?case
  proof (cases)
    assume s-in-b: s∈b
    assume Γ⊢⟨t3,Normal s⟩ =n⇒ t
    with Cond.hyps t3 noFault
    obtain Γ⊢⟨t1,Normal s⟩ =n⇒ t Γ⊢⟨t2,Normal s⟩ =n⇒ t
      by auto
    with s-in-b c2 show ?thesis
      by (auto intro: execn.intros)
  next
    assume s-notin-b: s∉b
    assume Γ⊢⟨e3,Normal s⟩ =n⇒ t
    with Cond.hyps e3 noFault
    obtain Γ⊢⟨e1,Normal s⟩ =n⇒ t Γ⊢⟨e2,Normal s⟩ =n⇒ t
      by auto
    with s-notin-b c2 show ?thesis
      by (auto intro: execn.intros)
  qed
next
  case (While b bdy1)
  have noFault: ¬ isFault t by fact
  have (While b bdy1 ∩g c2) = Some c by fact
  then obtain bdy2 bdy where
    c2: c2=While b bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=While b bdy
    by (auto simp add: inter-guards-While)
  have exec-c: Γ⊢⟨c,Normal s⟩ =n⇒ t by fact
  {
    fix s t n w w1 w2
    assume exec-w: Γ⊢⟨w,Normal s⟩ =n⇒ t
    assume w: w=While b bdy
    assume noFault: ¬ isFault t
    from exec-w w noFault
    have Γ⊢⟨While b bdy1,Normal s⟩ =n⇒ t ∧
          Γ⊢⟨While b bdy2,Normal s⟩ =n⇒ t
    proof (induct)
      prefer 10
      case (WhileTrue s b' bdy' n s' s'')
      have eqs: While b'  bdy' = While b bdy by fact
      from WhileTrue have s-in-b: s ∈ b by simp
      have noFault-s'': ¬ isFault s''  by fact
```

from *WhileTrue*
have *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ s\rangle =n\Rightarrow s'$ **by** *simp*
from *WhileTrue*
have *exec-w*: $\Gamma\vdash\langle While\ b\ bdy,s'\rangle =n\Rightarrow s''$ **by** *simp*
show *?case*
proof (*cases* s')
  case (*Fault f*)
  with *exec-w* **have** $s''=Fault\ f$
    by (*auto intro*: *execn-Fault-end*)
  with *noFault-s''* **show** *?thesis* **by** *simp*
next
  case (*Normal* s''')
  with *exec-bdy bdy While.hyps*
  obtain $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
        $\Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
    by *auto*
  moreover
  from *Normal WhileTrue*
  obtain
    $\Gamma\vdash\langle While\ b\ bdy1,Normal\ s'''\rangle =n\Rightarrow s''$
    $\Gamma\vdash\langle While\ b\ bdy2,Normal\ s'''\rangle =n\Rightarrow s''$
    by *simp*
  ultimately show *?thesis*
    using *s-in-b Normal*
    by (*auto intro*: *execn.intros*)
next
  case (*Abrupt* s''')
  with *exec-bdy bdy While.hyps*
  obtain $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'''$
        $\Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Abrupt\ s'''$
    by *auto*
  moreover
  from *Abrupt WhileTrue*
  obtain
    $\Gamma\vdash\langle While\ b\ bdy1,Abrupt\ s'''\rangle =n\Rightarrow s''$
    $\Gamma\vdash\langle While\ b\ bdy2,Abrupt\ s'''\rangle =n\Rightarrow s''$
    by *simp*
  ultimately show *?thesis*
    using *s-in-b Abrupt*
    by (*auto intro*: *execn.intros*)
next
  case *Stuck*
  with *exec-bdy bdy While.hyps*
  obtain $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Stuck$
        $\Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Stuck$
    by *auto*
  moreover
  from *Stuck WhileTrue*
  obtain

$\Gamma\vdash\langle$*While b bdy1*,*Stuck*$\rangle$ $=n\Rightarrow$ s''
$\Gamma\vdash\langle$*While b bdy2*,*Stuck*$\rangle$ $=n\Rightarrow$ s''
by *simp*
ultimately show *?thesis*
using *s-in-b Stuck*
by (*auto intro*: *execn.intros*)
qed
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.intros*)
qed (*simp-all*)
}
with *this* [*OF exec-c c noFault*] *c2*
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** (*simp add*: *inter-guards-Call*)
next
case (*DynCom f1*)
have *noFault*: $\neg$ *isFault t* **by** *fact*
have (*DynCom f1* $\cap_g$ *c2*) = *Some c* **by** *fact*
then obtain *f2 f* **where**
c2: *c2*=*DynCom f2* **and**
f-defined: $\forall$ *s*. ((*f1 s*) $\cap_g$ (*f2 s*)) $\neq$ *None* **and**
c: *c*=*DynCom* (λs. *the* ((*f1 s*) $\cap_g$ (*f2 s*)))
by (*auto simp add*: *inter-guards-DynCom*)
have $\Gamma\vdash\langle$*c*,*Normal s*$\rangle$ $=n\Rightarrow$ *t* **by** *fact*
with *c* **have** $\Gamma\vdash\langle$*DynCom* (λs. *the* ((*f1 s*) $\cap_g$ (*f2 s*))),*Normal s*$\rangle$ $=n\Rightarrow$ *t* **by** *simp*
then show *?case*
proof (*cases*)
assume *exec-f*: $\Gamma\vdash\langle$*the* (*f1 s* $\cap_g$ *f2 s*),*Normal s*$\rangle$ $=n\Rightarrow$ *t*
from *f-defined* **obtain** *f* **where** (*f1 s* $\cap_g$ *f2 s*) = *Some f*
by *auto*
with *DynCom.hyps this exec-f c2 noFault*
show *?thesis*
using *execn.DynCom* **by** *fastforce*
qed
next
case *Guard* **thus** *?case*
by (*fastforce elim*: *execn-Normal-elim-cases intro*: *execn.intros*
simp add: *inter-guards-Guard*)
next
case *Throw* **thus** *?case*
by (*fastforce elim*: *execn-Normal-elim-cases*
simp add: *inter-guards-Throw*)
next
case (*Catch a1 a2*)
have *noFault*: $\neg$ *isFault t* **by** *fact*
have (*Catch a1 a2* $\cap_g$ *c2*) = *Some c* **by** *fact*
then obtain *b1 b2 d1 d2* **where**

c2: *c2*=*Catch b1 b2* **and**
d1: (*a1* $\cap_g$ *b1*) = *Some d1* **and** *d2*: (*a2* $\cap_g$ *b2*) = *Some d2* **and**
c: *c*=*Catch d1 d2*
by (*auto simp add*: *inter-guards-Catch*)
have Γ⊢⟨*c*,*Normal s*⟩ =*n*⇒ *t* **by** *fact*
with *c* **have** Γ⊢⟨*Catch d1 d2*,*Normal s*⟩ =*n*⇒ *t* **by** *simp*
then show *?case*
proof (*cases*)
fix *s'*
assume Γ⊢⟨*d1*,*Normal s*⟩ =*n*⇒ *Abrupt s'*
with *d1 Catch.hyps*
obtain Γ⊢⟨*a1*,*Normal s*⟩ =*n*⇒ *Abrupt s'* **and** Γ⊢⟨*b1*,*Normal s*⟩ =*n*⇒ *Abrupt s'*
by *auto*
moreover
assume Γ⊢⟨*d2*,*Normal s'*⟩ =*n*⇒ *t*
with *d2 Catch.hyps noFault*
obtain Γ⊢⟨*a2*,*Normal s'*⟩ =*n*⇒ *t* **and** Γ⊢⟨*b2*,*Normal s'*⟩ =*n*⇒ *t*
by *auto*
ultimately
show *?thesis*
using *c2* **by** (*auto intro*: *execn.intros*)
next
assume ¬ *isAbr t*
moreover
assume Γ⊢⟨*d1*,*Normal s*⟩ =*n*⇒ *t*
with *d1 Catch.hyps noFault*
obtain Γ⊢⟨*a1*,*Normal s*⟩ =*n*⇒ *t* **and** Γ⊢⟨*b1*,*Normal s*⟩ =*n*⇒ *t*
by *auto*
ultimately
show *?thesis*
using *c2* **by** (*auto intro*: *execn.intros*)
qed
qed

lemma *inter-guards-execn-noFault*:
assumes *c*: (*c1* $\cap_g$ *c2*) = *Some c*
assumes *exec-c*: Γ⊢⟨*c*,*s*⟩ =*n*⇒ *t*
assumes *noFault*: ¬ *isFault t*
shows Γ⊢⟨*c1*,*s*⟩ =*n*⇒ *t* ∧ Γ⊢⟨*c2*,*s*⟩ =*n*⇒ *t*
proof (*cases s*)
case (*Fault f*)
with *exec-c* **have** *t* = *Fault f*
by (*auto intro*: *execn-Fault-end*)
with *noFault* **show** *?thesis*
by *simp*
next
case (*Abrupt s'*)

  with *exec-c* **have** t=*Abrupt* s'
    by (*simp add*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis* **by** *auto*
next
  case *Stuck*
  with *exec-c* **have** t=*Stuck*
    by (*simp add*: *execn-Stuck-end*)
  with *Stuck* **show** *?thesis* **by** *auto*
next
  case (*Normal* s')
  with *exec-c noFault inter-guards-execn-Normal-noFault* [*OF c*]
  show *?thesis*
    by *blast*
qed

lemma *inter-guards-exec-noFault*:
  assumes *c*: $(c1 \cap_g c2) = Some\ c$
  assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
  assumes *noFault*: $\neg\ isFault\ t$
  shows $\Gamma\vdash\langle c1,s\rangle \Rightarrow t \wedge \Gamma\vdash\langle c2,s\rangle \Rightarrow t$
proof −
  from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
    by (*auto simp add*: *exec-iff-execn*)
  from *c this noFault*
  have $\Gamma\vdash\langle c1,s\rangle =n\Rightarrow t \wedge \Gamma\vdash\langle c2,s\rangle =n\Rightarrow t$
    by (*rule inter-guards-execn-noFault*)
  thus *?thesis*
    by (*auto intro*: *execn-to-exec*)
qed

lemma *inter-guards-execn-Normal-Fault*:
  $\bigwedge c\ c2\ s\ n.\ [\![(c1 \cap_g c2) = Some\ c;\ \Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f]\!]$
    $\Longrightarrow (\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle c2,Normal\ s\rangle =n\Rightarrow Fault\ f)$
proof (*induct c1*)
  case *Skip* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Skip*)
next
  case (*Basic f*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Basic*)
next
  case (*Spec r*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Spec*)
next
  case (*Seq a1 a2*)
  have $(Seq\ a1\ a2 \cap_g c2) = Some\ c$ **by** *fact*
  then obtain *b1 b2 d1 d2* **where**
    c2: *c2=Seq b1 b2* **and**
    d1: $(a1 \cap_g b1) = Some\ d1$ **and** *d2*: $(a2 \cap_g b2) = Some\ d2$ **and**
    c: *c=Seq d1 d2*
    by (*auto simp add*: *inter-guards-Seq*)
  have $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*

```
  with c obtain s' where
    exec-d1: Γ⊢⟨d1,Normal s⟩ =n⇒ s' and
    exec-d2: Γ⊢⟨d2,s'⟩ =n⇒ Fault f
    by (auto elim: execn-Normal-elim-cases)
  show ?case
  proof (cases s')
    case (Fault f')
    with exec-d2 have f'=f
      by (auto dest: execn-Fault-end)
    with Fault d1 exec-d1
    have Γ⊢⟨a1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨b1,Normal s⟩ =n⇒ Fault f
      by (auto dest: Seq.hyps)
    thus ?thesis
    proof (cases rule: disjE [consumes 1])
      assume Γ⊢⟨a1,Normal s⟩ =n⇒ Fault f
      hence Γ⊢⟨Seq a1 a2,Normal s⟩ =n⇒ Fault f
        by (auto intro: execn.intros)
      thus ?thesis
        by simp
    next
      assume Γ⊢⟨b1,Normal s⟩ =n⇒ Fault f
      hence Γ⊢⟨Seq b1 b2,Normal s⟩ =n⇒ Fault f
        by (auto intro: execn.intros)
      with c2 show ?thesis
        by simp
    qed
  next
    case Abrupt with exec-d2 show ?thesis by (auto dest: execn-Abrupt-end)
  next
    case Stuck with exec-d2 show ?thesis by (auto dest: execn-Stuck-end)
  next
    case (Normal s'')
    with inter-guards-execn-noFault [OF d1 exec-d1] obtain
      exec-a1: Γ⊢⟨a1,Normal s⟩ =n⇒ Normal s'' and
      exec-b1: Γ⊢⟨b1,Normal s⟩ =n⇒ Normal s''
      by simp
    moreover from d2 exec-d2 Normal
    have Γ⊢⟨a2,Normal s''⟩ =n⇒ Fault f ∨ Γ⊢⟨b2,Normal s''⟩ =n⇒ Fault f
      by (auto dest: Seq.hyps)
    ultimately show ?thesis
      using c2 by (auto intro: execn.intros)
  qed
next
  case (Cond b t1 e1)
  have (Cond b t1 e1 ∩g c2) = Some c by fact
  then obtain t2 e2 t e where
    c2: c2=Cond b t2 e2 and
    t: (t1 ∩g t2) = Some t and
    e: (e1 ∩g e2) = Some e and
```

```
    c: c=Cond b t e
    by (auto simp add: inter-guards-Cond)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨Cond b t e,Normal s⟩ =n⇒ Fault f by simp
  thus ?case
  proof (cases)
    assume s ∈ b
    moreover assume Γ⊢⟨t,Normal s⟩ =n⇒ Fault f
    with t have Γ⊢⟨t1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨t2,Normal s⟩ =n⇒ Fault f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  next
    assume s ∉ b
    moreover assume Γ⊢⟨e,Normal s⟩ =n⇒ Fault f
    with e have Γ⊢⟨e1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨e2,Normal s⟩ =n⇒ Fault f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  qed
next
  case (While b bdy1)
  have (While b bdy1 ∩g c2) = Some c by fact
  then obtain bdy2 bdy where
    c2: c2=While b bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=While b bdy
    by (auto simp add: inter-guards-While)
  have exec-c: Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  {
    fix s t n w w1 w2
    assume exec-w: Γ⊢⟨w,Normal s⟩ =n⇒ t
    assume w: w=While b bdy
    assume Fault: t=Fault f
    from exec-w w Fault
    have Γ⊢⟨While b bdy1,Normal s⟩ =n⇒ Fault f∨
          Γ⊢⟨While b bdy2,Normal s⟩ =n⇒ Fault f
    proof (induct)
      case (WhileTrue s b' bdy' n  s' s'')
      have eqs: While b' bdy' = While b bdy by fact
      from WhileTrue have s-in-b: s ∈ b by simp
      have Fault-s'': s''=Fault f  by fact
      from WhileTrue
      have exec-bdy: Γ⊢⟨bdy,Normal s⟩ =n⇒ s' by simp
      from WhileTrue
      have exec-w: Γ⊢⟨While b bdy,s'⟩ =n⇒ s'' by simp
      show ?case
      proof (cases s')
        case (Fault f')
        with exec-w Fault-s'' have f'=f
          by (auto dest: execn-Fault-end)
```

```
        with Fault exec-bdy bdy While.hyps
        have Γ⊢⟨bdy1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨bdy2,Normal s⟩ =n⇒ Fault f
          by auto
        with s-in-b show ?thesis
          by (fastforce intro: execn.intros)
      next
        case (Normal s''')
        with inter-guards-execn-noFault [OF bdy exec-bdy]
        obtain Γ⊢⟨bdy1,Normal s⟩ =n⇒ Normal s'''
               Γ⊢⟨bdy2,Normal s⟩ =n⇒ Normal s'''
          by auto
        moreover
        from Normal WhileTrue
        have Γ⊢⟨While b bdy1,Normal s'''⟩ =n⇒ Fault f ∨
              Γ⊢⟨While b bdy2,Normal s'''⟩ =n⇒ Fault f
          by simp
        ultimately show ?thesis
          using s-in-b by (fastforce intro: execn.intros)
      next
        case (Abrupt s''')
        with exec-w Fault-s'' show ?thesis by (fastforce dest: execn-Abrupt-end)
      next
        case Stuck
        with exec-w Fault-s'' show ?thesis by (fastforce dest: execn-Stuck-end)
      qed
    next
      case WhileFalse thus ?case by (auto intro: execn.intros)
    qed (simp-all)
  }
  with this [OF exec-c c] c2
  show ?case
    by auto
next
  case Call thus ?case by (fastforce simp add: inter-guards-Call)
next
  case (DynCom f1)
  have (DynCom f1 ∩g c2) = Some c by fact
  then obtain f2  where
    c2: c2=DynCom f2 and
    F-defined: ∀ s. ((f1 s) ∩g (f2 s)) ≠ None and
    c: c=DynCom (λs. the ((f1 s) ∩g (f2 s)))
    by (auto simp add: inter-guards-DynCom)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨DynCom (λs. the ((f1 s) ∩g (f2 s))),Normal s⟩ =n⇒ Fault f
by simp
  then show ?case
  proof (cases)
    assume exec-F: Γ⊢⟨the (f1 s ∩g f2 s),Normal s⟩ =n⇒ Fault f
    from F-defined obtain F where (f1 s ∩g f2 s) = Some F
```

```
      by auto
    with DynCom.hyps this exec-F c2
    show ?thesis
      by (fastforce intro: execn.intros)
  qed
next
  case (Guard m g1 bdy1)
  have (Guard m g1 bdy1 ∩g c2) = Some c by fact
  then obtain g2 bdy2 bdy where
    c2: c2=Guard m g2 bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=Guard m (g1 ∩ g2) bdy
    by (auto simp add: inter-guards-Guard)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨Guard m (g1 ∩ g2) bdy,Normal s⟩ =n⇒ Fault f
    by simp
  thus ?case
  proof (cases)
    assume f-m: Fault f = Fault m
    assume s ∉ g1 ∩ g2
    hence s∉g1 ∨ s∉g2
      by blast
    with c2 f-m show ?thesis
      by (auto intro: execn.intros)
  next
    assume s ∈ g1 ∩ g2
    moreover
    assume Γ⊢⟨bdy,Normal s⟩ =n⇒ Fault f
    with bdy have Γ⊢⟨bdy1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨bdy2,Normal s⟩ =n⇒
Fault f
      by (rule Guard.hyps)
    ultimately show ?thesis
      using c2
      by (auto intro: execn.intros)
  qed
next
  case Throw thus ?case by (fastforce simp add: inter-guards-Throw)
next
  case (Catch a1 a2)
  have (Catch a1 a2 ∩g c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Catch b1 b2 and
    d1: (a1 ∩g b1) = Some d1 and d2: (a2 ∩g b2) = Some d2 and
    c: c=Catch d1 d2
    by (auto simp add: inter-guards-Catch)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨Catch d1 d2,Normal s⟩ =n⇒ Fault f by simp
  thus ?case
  proof (cases)
```

fix *s'*
assume Γ⊢⟨*d1*,*Normal s*⟩ =*n*⇒ *Abrupt s'*
from *inter-guards-execn-noFault* [*OF d1 this*] **obtain**
exec-a1: Γ⊢⟨*a1*,*Normal s*⟩ =*n*⇒ *Abrupt s'* **and**
exec-b1: Γ⊢⟨*b1*,*Normal s*⟩ =*n*⇒ *Abrupt s'*
by *simp*
moreover assume Γ⊢⟨*d2*,*Normal s'*⟩ =*n*⇒ *Fault f*
with *d2*
have Γ⊢⟨*a2*,*Normal s'*⟩ =*n*⇒ *Fault f* ∨ Γ⊢⟨*b2*,*Normal s'*⟩ =*n*⇒ *Fault f*
by (*auto dest*: *Catch.hyps*)
ultimately show *?thesis*
using *c2* **by** (*fastforce intro*: *execn.intros*)
next
assume Γ⊢⟨*d1*,*Normal s*⟩ =*n*⇒ *Fault f*
with *d1* **have** Γ⊢⟨*a1*,*Normal s*⟩ =*n*⇒ *Fault f* ∨ Γ⊢⟨*b1*,*Normal s*⟩ =*n*⇒ *Fault f*
by (*auto dest*: *Catch.hyps*)
with *c2* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed

lemma *inter-guards-execn-Fault*:
assumes *c*: (*c1* $\cap_g$ *c2*) = *Some c*
assumes *exec-c*: Γ⊢⟨*c*,*s*⟩ =*n*⇒ *Fault f*
shows Γ⊢⟨*c1*,*s*⟩ =*n*⇒ *Fault f* ∨ Γ⊢⟨*c2*,*s*⟩ =*n*⇒ *Fault f*
proof (*cases s*)
case (*Fault f*)
with *exec-c* **show** *?thesis*
by (*auto dest*: *execn-Fault-end*)
next
case (*Abrupt s'*)
with *exec-c* **show** *?thesis*
by (*fastforce dest*: *execn-Abrupt-end*)
next
case *Stuck*
with *exec-c* **show** *?thesis*
by (*fastforce dest*: *execn-Stuck-end*)
next
case (*Normal s'*)
with *exec-c inter-guards-execn-Normal-Fault* [*OF c*]
show *?thesis*
by *blast*
qed

lemma *inter-guards-exec-Fault*:
assumes *c*: (*c1* $\cap_g$ *c2*) = *Some c*
assumes *exec-c*: Γ⊢⟨*c*,*s*⟩ ⇒ *Fault f*

shows $\Gamma\vdash\langle c1,s\rangle \Rightarrow$ *Fault f* $\vee$ $\Gamma\vdash\langle c2,s\rangle \Rightarrow$ *Fault f*
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash\langle c,s\rangle =n\Rightarrow$ *Fault f*
by (*auto simp add*: *exec-iff-execn*)
from *c this*
have $\Gamma\vdash\langle c1,s\rangle =n\Rightarrow$ *Fault f* $\vee$ $\Gamma\vdash\langle c2,s\rangle =n\Rightarrow$ *Fault f*
by (*rule inter-guards-execn-Fault*)
thus *?thesis*
by (*auto intro*: *execn-to-exec*)
qed

2.9 Restriction of Procedure Environment

lemma *restrict-SomeD*: $(m|_A)\ x = Some\ y \Longrightarrow m\ x = Some\ y$
by (*auto simp add*: *restrict-map-def split*: *if-split-asm*)

lemma *restrict-dom-same* [*simp*]: $m|_{dom\ m} = m$
apply (*rule ext*)
apply (*clarsimp simp add*: *restrict-map-def*)
apply (*simp only*: *not-None-eq* [*symmetric*])
apply *rule*
apply (*drule sym*)
apply *blast*
done

lemma *restrict-in-dom*: $x \in A \Longrightarrow (m|_A)\ x = m\ x$
by (*auto simp add*: *restrict-map-def*)

lemma *exec-restrict-to-exec*:
assumes *exec-restrict*: $\Gamma|_A\vdash\langle c,s\rangle \Rightarrow t$
assumes *notStuck*: $t\neq Stuck$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *exec-restrict notStuck*
by (*induct*) (*auto intro*: *exec.intros dest*: *restrict-SomeD Stuck-end*)

lemma *execn-restrict-to-execn*:
assumes *exec-restrict*: $\Gamma|_A\vdash\langle c,s\rangle =n\Rightarrow t$
assumes *notStuck*: $t\neq Stuck$
shows $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
using *exec-restrict notStuck*
by (*induct*) (*auto intro*: *execn.intros dest*: *restrict-SomeD execn-Stuck-end*)

lemma *restrict-NoneD*: $m\ x = None \Longrightarrow (m|_A)\ x = None$
by (*auto simp add*: *restrict-map-def split*: *if-split-asm*)

lemma *execn-to-execn-restrict*:
assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$

shows $\exists\, t'.\ \Gamma|_P\vdash\langle c,s\rangle =n\Rightarrow\ t' \wedge (t{=}Stuck \longrightarrow t'{=}Stuck) \wedge$
$(\forall f.\ t{=}Fault\, f \longrightarrow t'{\in}\{Fault\, f, Stuck\}) \wedge (t'{\neq}Stuck \longrightarrow t'{=}t)$
using *execn*
proof (*induct*)
  case *Skip* **show** *?case* **by** (*blast intro*: *execn.Skip*)
next
  case *Guard* **thus** *?case* **by** (*auto intro*: *execn.Guard*)
next
  case *GuardFault* **thus** *?case* **by** (*auto intro*: *execn.GuardFault*)
next
  case *FaultProp* **thus** *?case* **by** (*auto intro*: *execn.FaultProp*)
next
  case *Basic* **thus** *?case* **by** (*auto intro*: *execn.Basic*)
next
  case *Spec* **thus** *?case* **by** (*auto intro*: *execn.Spec*)
next
  case *SpecStuck* **thus** *?case* **by** (*auto intro*: *execn.SpecStuck*)
next
  case *Seq* **thus** *?case* **by** (*metis insertCI execn.Seq StuckProp*)
next
  case *CondTrue* **thus** *?case* **by** (*auto intro*: *execn.CondTrue*)
next
  case *CondFalse* **thus** *?case* **by** (*auto intro*: *execn.CondFalse*)
next
  case *WhileTrue* **thus** *?case* **by** (*metis insertCI execn.WhileTrue StuckProp*)
next
  case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.WhileFalse*)
next
  case (*Call p bdy n s s'*)
  have $\Gamma\ p = Some\ bdy$ **by** *fact*
  show *?case*
  proof (*cases* $p \in P$)
    case *True*
    with *Call* **have** $(\Gamma|_P)\ p = Some\ bdy$
      by (*simp*)
    with *Call* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case *False*
    hence $(\Gamma|_P)\ p = None$ **by** *simp*
    thus *?thesis*
      by (*auto intro*: *execn.CallUndefined*)
  qed
next
  case (*CallUndefined p n s*)
  have $\Gamma\ p = None$ **by** *fact*
  hence $(\Gamma|_P)\ p = None$ **by** (*rule restrict-NoneD*)
  thus *?case* **by** (*auto intro*: *execn.CallUndefined*)
next

case *StuckProp* **thus** *?case* **by** (*auto intro*: *execn.StuckProp*)
next
case *DynCom* **thus** *?case* **by** (*auto intro*: *execn.DynCom*)
next
case *Throw* **thus** *?case* **by** (*auto intro*: *execn.Throw*)
next
case *AbruptProp* **thus** *?case* **by** (*auto intro*: *execn.AbruptProp*)
next
case (*CatchMatch c1 s n s′ c2 s′′*)
from *CatchMatch.hyps*
obtain t' t'' **where**
exec-res-c1: $\Gamma|_P\vdash\langle c1,Normal\ s\rangle\ =n\Rightarrow\ t'$ **and**
t′-notStuck: $t' \neq Stuck \longrightarrow t' = Abrupt\ s'$ **and**
exec-res-c2: $\Gamma|_P\vdash\langle c2,Normal\ s'\rangle\ =n\Rightarrow\ t''$ **and**
s′′-Stuck: $s'' = Stuck \longrightarrow t'' = Stuck$ **and**
s′′-Fault: $\forall f.\ s'' = Fault\ f \longrightarrow t'' \in \{Fault\ f,\ Stuck\}$ **and**
t′′-notStuck: $t'' \neq Stuck \longrightarrow t'' = s''$
by *auto*
show *?case*
proof (*cases* t'=*Stuck*)
case *True*
with *exec-res-c1*
have $\Gamma|_P\vdash\langle Catch\ c1\ c2,Normal\ s\rangle\ =n\Rightarrow\ Stuck$
by (*auto intro*: *execn.CatchMiss*)
thus *?thesis*
by *auto*
next
case *False*
with *t′-notStuck* **have** t'= *Abrupt* s'
by *simp*
with *exec-res-c1 exec-res-c2*
have $\Gamma|_P\vdash\langle Catch\ c1\ c2,Normal\ s\rangle\ =n\Rightarrow\ t''$
by (*auto intro*: *execn.CatchMatch*)
with *s′′-Stuck s′′-Fault t′′-notStuck*
show *?thesis*
by *blast*
qed
next
case (*CatchMiss c1 s n w c2*)
have *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s\rangle\ =n\Rightarrow\ w$ **by** *fact*
from *CatchMiss.hyps* **obtain** w' **where**
exec-c1′: $\Gamma|_P\vdash\langle c1,Normal\ s\rangle\ =n\Rightarrow\ w'$ **and**
w-Stuck: $w = Stuck \longrightarrow w' = Stuck$ **and**
w-Fault: $\forall f.\ w = Fault\ f \longrightarrow w' \in \{Fault\ f,\ Stuck\}$ **and**
w′-noStuck: $w' \neq Stuck \longrightarrow w' = w$
by *auto*
have *noAbr-w*: $\neg$ *isAbr w* **by** *fact*
show *?case*
proof (*cases* w')

case (*Normal s'*)
with *w'-noStuck* **have** *w'=w*
by *simp*
with *exec-c1' Normal w-Stuck w-Fault w'-noStuck*
show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
next
case (*Abrupt s'*)
with *w'-noStuck* **have** *w'=w*
by *simp*
with *noAbr-w Abrupt* **show** *?thesis* **by** *simp*
next
case (*Fault f*)
with *w'-noStuck* **have** *w'=w*
by *simp*
with *exec-c1' Fault w-Stuck w-Fault w'-noStuck*
show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
next
case *Stuck*
with *exec-c1' w-Stuck w-Fault w'-noStuck*
show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
qed
qed

lemma *exec-to-exec-restrict*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
shows $\exists\, t'.\ \Gamma|_P\vdash\langle c,s\rangle \Rightarrow t' \wedge (t{=}Stuck \longrightarrow t'{=}Stuck) \wedge$
$(\forall f.\ t{=}Fault\ f \longrightarrow t'{\in}\{Fault\ f,Stuck\}) \wedge (t'{\neq}Stuck \longrightarrow t'{=}t)$
proof −
from *exec* **obtain** *n* **where**
execn-strip: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *execn-to-execn-restrict* [**where** *P=P*,*OF this*]
obtain *t'* **where**
$\Gamma|_P\vdash\langle c,s\rangle =n\Rightarrow t'$
$t{=}Stuck \longrightarrow t'{=}Stuck$ $\forall f.\ t{=}Fault\ f \longrightarrow t'{\in}\{Fault\ f,Stuck\}$ $t'{\neq}Stuck \longrightarrow t'{=}t$
by *blast*
thus *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *notStuck-GuardD*:
$[\![\Gamma\vdash\langle Guard\ m\ g\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in g]\!] \Longrightarrow \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Guard*)

lemma *notStuck-SeqD1*:

$[\![\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-SeqD2*:
$[\![\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow s']\!] \Longrightarrow \Gamma\vdash\langle c2,s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-SeqD*:
$[\![\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow$
$\Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\} \wedge (\forall s'.\ \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash\langle c2,s'\rangle \Rightarrow\notin\{Stuck\})$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-CondTrueD*:
$[\![\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b]\!] \Longrightarrow \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CondTrue*)

lemma *notStuck-CondFalseD*:
$[\![\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \notin b]\!] \Longrightarrow \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CondFalse*)

lemma *notStuck-WhileTrueD1*:
$[\![\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b]\!]$
$\Longrightarrow \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.WhileTrue*)

lemma *notStuck-WhileTrueD2*:
$[\![\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow s';\ s \in b]\!]$
$\Longrightarrow \Gamma\vdash\langle While\ b\ c,s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.WhileTrue*)

lemma *notStuck-CallD*:
$[\![\Gamma\vdash\langle Call\ p\ ,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\ p = Some\ bdy]\!]$
$\Longrightarrow \Gamma\vdash\langle bdy,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Call*)

lemma *notStuck-CallDefinedD*:
$[\![\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!]$
$\Longrightarrow \Gamma\ p \neq None$
by (*cases* Γ *p*)
(*auto simp add*: *final-notin-def dest*: *exec.CallUndefined*)

lemma *notStuck-DynComD*:
$[\![\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!]$
$\Longrightarrow \Gamma\vdash\langle (c\ s),Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.DynCom*)

lemma *notStuck-CatchD1*:
 $[\![\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
 by (*auto simp add*: *final-notin-def dest*: *exec.CatchMatch exec.CatchMiss*)

lemma *notStuck-CatchD2*:
 $[\![\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow Abrupt\ s']\!]$
 $\Longrightarrow \Gamma\vdash\langle c2,Normal\ s'\rangle \Rightarrow\notin\{Stuck\}$
 by (*auto simp add*: *final-notin-def dest*: *exec.CatchMatch*)

2.10 Miscellaneous

lemma *execn-noguards-no-Fault*:
 assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
 assumes *noguards-c*: *noguards c*
 assumes *noguards*-Γ: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
 assumes *s-no-Fault*: $\neg$ *isFault s*
 shows $\neg$ *isFault t*
  using *execn noguards-c s-no-Fault*
  proof (*induct*)
   case (*Call p bdy n s t*) **with** *noguards*-Γ **show** *?case*
    apply −
    apply (*drule bspec* [**where** *x*=*p*])
    apply *auto*
    done
  qed (*auto*)

lemma *exec-noguards-no-Fault*:
 assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
 assumes *noguards-c*: *noguards c*
 assumes *noguards*-Γ: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
 assumes *s-no-Fault*: $\neg$ *isFault s*
 shows $\neg$ *isFault t*
  using *exec noguards-c s-no-Fault*
  proof (*induct*)
   case (*Call p bdy s t*) **with** *noguards*-Γ **show** *?case*
    apply −
    apply (*drule bspec* [**where** *x*=*p*])
    apply *auto*
    done
  qed *auto*

lemma *execn-nothrows-no-Abrupt*:
 assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
 assumes *nothrows-c*: *nothrows c*
 assumes *nothrows*-Γ: $\forall p \in dom\ \Gamma.\ nothrows\ (the\ (\Gamma\ p))$
 assumes *s-no-Abrupt*: $\neg$(*isAbr s*)
 shows $\neg$(*isAbr t*)
  using *execn nothrows-c s-no-Abrupt*
  proof (*induct*)

case (*Call p bdy n s t*) **with** *nothrows*-Γ **show** *?case*
apply −
apply (*drule bspec* [**where** *x*=*p*])
apply *auto*
done
qed (*auto*)

lemma *exec-nothrows-no-Abrupt*:
assumes *exec*: Γ⊢⟨*c*,*s*⟩ ⇒ *t*
assumes *nothrows-c*: *nothrows c*
assumes *nothrows*-Γ: ∀ *p* ∈ *dom* Γ. *nothrows* (*the* (Γ *p*))
assumes *s-no-Abrupt*: ¬(*isAbr s*)
shows ¬(*isAbr t*)
using *exec nothrows-c s-no-Abrupt*
proof (*induct*)
case (*Call p bdy s t*) **with** *nothrows*-Γ **show** *?case*
apply −
apply (*drule bspec* [**where** *x*=*p*])
apply *auto*
done
qed (*auto*)

end

3 Terminating Programs

theory *Termination* **imports** *Semantic* **begin**

3.1 Inductive Characterisation: Γ⊢*c*↓*s*

inductive *terminates*::(′*s*,′*p*,′*f*) *body* ⇒ (′*s*,′*p*,′*f*) *com* ⇒ (′*s*,′*f*) *xstate* ⇒ *bool*
(-⊢- ↓ - [*60*,*20*,*60*] *89*)
for Γ::(′*s*,′*p*,′*f*) *body*
where
Skip: Γ⊢*Skip* ↓(*Normal s*)

| *Basic*: Γ⊢*Basic f* ↓(*Normal s*)

| *Spec*: Γ⊢*Spec r* ↓(*Normal s*)

| *Guard*: ⟦*s*∈*g*; Γ⊢*c*↓(*Normal s*)⟧
⟹
Γ⊢*Guard f g c*↓(*Normal s*)

| *GuardFault*: *s*∉*g*
⟹
Γ⊢*Guard f g c*↓(*Normal s*)

| *Fault* [*intro*,*simp*]: $\Gamma\vdash c\downarrow Fault\ f$

| *Seq*: $[\![\Gamma\vdash c_1\downarrow Normal\ s;\ \forall s'.\ \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash c_2\downarrow s']\!]$
$\Longrightarrow$
$\Gamma\vdash Seq\ c_1\ \ c_2\downarrow(Normal\ s)$

| *CondTrue*: $[\![s \in b;\ \Gamma\vdash c_1\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash Cond\ b\ c_1\ \ c_2\downarrow(Normal\ s)$

| *CondFalse*: $[\![s \notin b;\ \Gamma\vdash c_2\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash Cond\ b\ c_1\ \ c_2\downarrow(Normal\ s)$

| *WhileTrue*: $[\![s \in b;\ \Gamma\vdash c\downarrow(Normal\ s);$
$\forall s'.\ \Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash While\ b\ c\downarrow s']\!]$
$\Longrightarrow$
$\Gamma\vdash While\ b\ c\downarrow(Normal\ s)$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash While\ b\ c\downarrow(Normal\ s)$

| *Call*: $[\![\Gamma\ p{=}Some\ bdy;\Gamma\vdash bdy\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash Call\ p\downarrow(Normal\ s)$

| *CallUndefined*: $[\![\Gamma\ p = None]\!]$
$\Longrightarrow$
$\Gamma\vdash Call\ p\downarrow(Normal\ s)$

| *Stuck* [*intro*,*simp*]: $\Gamma\vdash c\downarrow Stuck$

| *DynCom*: $[\![\Gamma\vdash(c\ s)\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash DynCom\ c\downarrow(Normal\ s)$

| *Throw*: $\Gamma\vdash Throw\downarrow(Normal\ s)$

| *Abrupt* [*intro*,*simp*]: $\Gamma\vdash c\downarrow Abrupt\ s$

| *Catch*: $[\![\Gamma\vdash c_1\downarrow Normal\ s;$
$\forall s'.\ \Gamma\vdash\langle c_1,Normal\ s\ \rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash c_2\downarrow Normal\ s']\!]$
$\Longrightarrow$
$\Gamma\vdash Catch\ c_1\ \ c_2\downarrow Normal\ s$

inductive-cases *terminates-elim-cases* [*cases set*]:
$\Gamma\vdash Skip \downarrow s$
$\Gamma\vdash Guard\ f\ g\ c \downarrow s$
$\Gamma\vdash Basic\ f \downarrow s$
$\Gamma\vdash Spec\ r \downarrow s$
$\Gamma\vdash Seq\ c1\ c2 \downarrow s$
$\Gamma\vdash Cond\ b\ c1\ c2 \downarrow s$
$\Gamma\vdash While\ b\ c \downarrow s$
$\Gamma\vdash Call\ p \downarrow s$
$\Gamma\vdash DynCom\ c \downarrow s$
$\Gamma\vdash Throw \downarrow s$
$\Gamma\vdash Catch\ c1\ c2 \downarrow s$

inductive-cases *terminates-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash Skip \downarrow Normal\ s$
$\Gamma\vdash Guard\ f\ g\ c \downarrow Normal\ s$
$\Gamma\vdash Basic\ f \downarrow Normal\ s$
$\Gamma\vdash Spec\ r \downarrow Normal\ s$
$\Gamma\vdash Seq\ c1\ c2 \downarrow Normal\ s$
$\Gamma\vdash Cond\ b\ c1\ c2 \downarrow Normal\ s$
$\Gamma\vdash While\ b\ c \downarrow Normal\ s$
$\Gamma\vdash Call\ p \downarrow Normal\ s$
$\Gamma\vdash DynCom\ c \downarrow Normal\ s$
$\Gamma\vdash Throw \downarrow Normal\ s$
$\Gamma\vdash Catch\ c1\ c2 \downarrow Normal\ s$

lemma *terminates-Skip'*: $\Gamma\vdash Skip \downarrow s$
by (*cases s*) (*auto intro*: *terminates.intros*)

lemma *terminates-Call-body*:
$\Gamma\ p{=}Some\ bdy \Longrightarrow \Gamma\vdash Call\ \ p \downarrow s = \Gamma\vdash(the\ (\Gamma\ p))\downarrow s$
by (*cases s*)
(*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*)

lemma *terminates-Normal-Call-body*:
$p \in dom\ \Gamma \Longrightarrow$
$\Gamma\vdash Call\ p \downarrow Normal\ s = \Gamma\vdash(the\ (\Gamma\ p))\downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*)

lemma *terminates-implies-exec*:
assumes *terminates*: $\Gamma\vdash c\downarrow s$
shows $\exists\, t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *terminates*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*iprover intro*: *exec.intros*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *exec.intros*)
next

```
  case (Spec r s) thus ?case
    by (cases ∃ t. (s,t)∈ r) (auto intro: exec.intros)
next
  case Guard thus ?case by (iprover intro: exec.intros)
next
  case GuardFault thus ?case by (iprover intro: exec.intros)
next
  case Fault thus ?case by (iprover intro: exec.intros)
next
  case Seq thus ?case by (iprover intro: exec-Seq')
next
  case CondTrue thus ?case by (iprover intro: exec.intros)
next
  case CondFalse thus ?case by (iprover intro: exec.intros)
next
  case WhileTrue thus ?case by (iprover intro: exec.intros)
next
  case WhileFalse thus ?case by (iprover intro: exec.intros)
next
  case (Call p bdy s)
  then obtain s' where
    Γ⊢⟨bdy,Normal s ⟩ ⇒ s'
    by iprover
  moreover have Γ p = Some bdy by fact
  ultimately show ?case
    by (cases s') (iprover intro: exec.intros)+
next
  case CallUndefined thus ?case by (iprover intro: exec.intros)
next
  case Stuck thus ?case by (iprover intro: exec.intros)
next
  case DynCom thus ?case by (iprover intro: exec.intros)
next
  case Throw thus ?case by (iprover intro: exec.intros)
next
  case Abrupt thus ?case by (iprover intro: exec.intros)
next
  case (Catch c1 s c2)
  then obtain s' where exec-c1: Γ⊢⟨c1,Normal s ⟩ ⇒ s'
    by iprover
  thus ?case
  proof (cases s')
    case (Normal s'')
    with exec-c1 show ?thesis by (auto intro!: exec.intros)
  next
    case (Abrupt s'')
    with exec-c1 Catch.hyps
    obtain t where Γ⊢⟨c2,Normal s'' ⟩ ⇒ t
      by auto
```

with *exec-c1 Abrupt* **show** *?thesis* **by** (*auto intro*: *exec.intros*)
next
case *Fault*
with *exec-c1* **show** *?thesis* **by** (*auto intro*!: *exec.CatchMiss*)
next
case *Stuck*
with *exec-c1* **show** *?thesis* **by** (*auto intro*!: *exec.CatchMiss*)
qed
qed

lemma *terminates-block*:
$[\![\Gamma\vdash bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)]\!]$
$\Longrightarrow \Gamma\vdash block\ init\ bdy\ return\ c \downarrow Normal\ s$
apply (*unfold block-def*)
apply (*fastforce intro*: *terminates.intros elim*!: *exec-Normal-elim-cases*
dest!: *not-isAbrD*)
done

lemma *terminates-block-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash block\ init\ bdy\ return\ c \downarrow Normal\ s$
assumes *e*: $[\![\Gamma\vdash bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s$
$t)$
$]\!] \Longrightarrow P$
shows *P*
proof −
have $\Gamma\vdash\langle Basic\ init, Normal\ s\rangle \Rightarrow Normal\ (init\ s)$
by (*auto intro*: *exec.intros*)
with *termi*
have $\Gamma\vdash bdy \downarrow Normal\ (init\ s)$
apply (*unfold block-def*)
apply (*elim terminates-Normal-elim-cases*)
by *simp*
moreover
{
fix *t*
assume *exec-bdy*: $\Gamma\vdash\langle bdy, Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
have $\Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$
proof −
from *exec-bdy*
have $\Gamma\vdash\langle Catch\ (Seq\ (Basic\ init)\ bdy)$
$(Seq\ (Basic\ (return\ s))\ Throw), Normal\ s\rangle \Rightarrow Normal\ t$
by (*fastforce intro*: *exec.intros*)
with *termi* **have** $\Gamma\vdash DynCom\ (\lambda t.\ Seq\ (Basic\ (return\ s))\ (c\ s\ t)) \downarrow Normal\ t$
apply (*unfold block-def*)
apply (*elim terminates-Normal-elim-cases*)
by *simp*
thus *?thesis*

apply (*elim terminates-Normal-elim-cases*)
apply (*auto intro*: *exec.intros*)
done
qed
}
ultimately show P **by** (*iprover intro*: *e*)
qed

lemma *terminates-call*:
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)]\!]$
$\Longrightarrow \Gamma\vdash call\ init\ p\ return\ c \downarrow Normal\ s$
apply (*unfold call-def*)
apply (*rule terminates-block*)
apply (*iprover intro*: *terminates.intros*)
apply (*auto elim*: *exec-Normal-elim-cases*)
done

lemma *terminates-callUndefined*:
$[\![\Gamma\ p = None]\!]$
$\Longrightarrow \Gamma\vdash call\ init\ p\ return\ result \downarrow Normal\ s$
apply (*unfold call-def*)
apply (*rule terminates-block*)
apply (*iprover intro*: *terminates.intros*)
apply (*auto elim*: *exec-Normal-elim-cases*)
done

lemma *terminates-call-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash call\ init\ p\ return\ c \downarrow Normal\ s$
assumes *bdy*: $\bigwedge bdy.\ [\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)]\!]$
$\Longrightarrow P$
assumes *undef*: $[\![\Gamma\ p = None]\!] \Longrightarrow P$
shows P
apply (*cases* Γ *p*)
apply (*erule undef*)
using *termi*
apply (*unfold call-def*)
apply (*erule terminates-block-elim*)
apply (*erule terminates-Normal-elim-cases*)
apply *simp*
apply (*frule* (*1*) *bdy*)
apply (*fastforce intro*: *exec.intros*)
apply *assumption*
apply *simp*
done

lemma *terminates-dynCall*:

$[\![\Gamma\vdash$*call init* $(p\ s)$ *return c* $\downarrow$ *Normal s*$]\!]$
$\Longrightarrow \Gamma\vdash$*dynCall init p return c* $\downarrow$ *Normal s*
apply (*unfold dynCall-def*)
apply (*auto intro*: *terminates.intros terminates-call*)
done

lemma *terminates-dynCall-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash$*dynCall init p return c* $\downarrow$ *Normal s*
assumes $[\![\Gamma\vdash$*call init* $(p\ s)$ *return c* $\downarrow$ *Normal s*$]\!] \Longrightarrow P$
shows P
using *termi*
apply (*unfold dynCall-def*)
apply (*elim terminates-Normal-elim-cases*)
apply *fact*
done

3.2 Lemmas about *sequence*, *flatten* and *Language.normalize*

lemma *terminates-sequence-app*:
$\bigwedge s$. $[\![\Gamma\vdash$*sequence Seq xs* $\downarrow$ *Normal s*;
$\forall s'$. $\Gamma\vdash\langle$*sequence Seq xs*,*Normal s* $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash$*sequence Seq ys* $\downarrow s']\!]$
$\Longrightarrow \Gamma\vdash$*sequence Seq* $(xs\ @\ ys) \downarrow$ *Normal s*
proof (*induct xs*)
case *Nil*
thus *?case* **by** (*auto intro*: *exec.intros*)
next
case (*Cons x xs*)
have *termi-x-xs*: $\Gamma\vdash$*sequence Seq* $(x\ \#\ xs) \downarrow$ *Normal s* **by** *fact*
have *termi-ys*: $\forall s'$. $\Gamma\vdash\langle$*sequence Seq* $(x\ \#\ xs)$,*Normal s* $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash$*sequence Seq ys* $\downarrow s'$ **by** *fact*
show *?case*
proof (*cases xs*)
case *Nil*
with *termi-x-xs termi-ys* **show** *?thesis*
by (*cases ys*) (*auto intro*: *terminates.intros*)
next
case *Cons*
from *termi-x-xs Cons*
have $\Gamma\vdash x \downarrow$ *Normal s*
by (*auto elim*: *terminates-Normal-elim-cases*)
moreover
{
fix s'
assume *exec-x*: $\Gamma\vdash\langle x$,*Normal s* $\rangle \Rightarrow s'$
have $\Gamma\vdash$*sequence Seq* $(xs\ @\ ys) \downarrow s'$
proof −
from *exec-x termi-x-xs Cons*
have *termi-xs*: $\Gamma\vdash$*sequence Seq xs* $\downarrow s'$
by (*auto elim*: *terminates-Normal-elim-cases*)

```
        show ?thesis
        proof (cases s')
          case (Normal s'')
          with exec-x termi-ys Cons
          have ∀ s'. Γ⊢⟨sequence Seq xs,Normal s'' ⟩ ⇒ s' ⟶ Γ⊢sequence Seq ys ↓
s'
            by (auto intro: exec.intros)
          from Cons.hyps [OF termi-xs [simplified Normal] this]
          have Γ⊢sequence Seq (xs @ ys) ↓ Normal s''.
          with Normal show ?thesis by simp
        next
          case Abrupt thus ?thesis by (auto intro: terminates.intros)
        next
          case Fault thus ?thesis by (auto intro: terminates.intros)
        next
          case Stuck thus ?thesis by (auto intro: terminates.intros)
        qed
      qed
    }
    ultimately show ?thesis
      using Cons
      by (auto intro: terminates.intros)
  qed
qed

lemma terminates-sequence-appD:
  ⋀s. Γ⊢sequence Seq (xs @ ys) ↓ Normal s
   ⟹ Γ⊢sequence Seq xs ↓ Normal s ∧
       (∀ s'. Γ⊢⟨sequence Seq xs,Normal s ⟩ ⇒ s' ⟶  Γ⊢sequence Seq ys ↓ s')
proof (induct xs)
  case Nil
  thus ?case
    by (auto elim: terminates-Normal-elim-cases exec-Normal-elim-cases
         intro: terminates.intros)
next
  case (Cons x xs)
  have termi-x-xs-ys: Γ⊢sequence Seq ((x # xs) @ ys) ↓ Normal s by fact
  show ?case
  proof (cases xs)
    case Nil
    with termi-x-xs-ys show ?thesis
      by (cases ys)
         (auto elim: terminates-Normal-elim-cases exec-Normal-elim-cases
           intro:  terminates-Skip')
  next
    case Cons
    with termi-x-xs-ys
    obtain termi-x: Γ⊢x ↓ Normal s and
          termi-xs-ys: ∀ s'. Γ⊢⟨x,Normal s ⟩ ⇒ s' ⟶  Γ⊢sequence Seq (xs@ys) ↓ s'
```

```
  by (auto elim: terminates-Normal-elim-cases)

have Γ⊢Seq x (sequence Seq xs) ↓ Normal s
proof (rule terminates.Seq [rule-format])
  show Γ⊢x ↓ Normal s by (rule termi-x)
next
  fix s'
  assume exec-x: Γ⊢⟨x,Normal s ⟩ ⇒ s'
  show Γ⊢sequence Seq xs ↓ s'
  proof −
    from termi-xs-ys [rule-format, OF exec-x]
    have termi-xs-ys': Γ⊢sequence Seq (xs@ys) ↓ s' .
    show ?thesis
    proof (cases s')
      case (Normal s'')
      from Cons.hyps [OF termi-xs-ys' [simplified Normal]]
      show ?thesis
        using Normal by auto
    next
      case Abrupt thus ?thesis by (auto intro: terminates.intros)
    next
      case Fault thus ?thesis by (auto intro: terminates.intros)
    next
      case Stuck thus ?thesis by (auto intro: terminates.intros)
    qed
  qed
qed
moreover
{
  fix s'
  assume exec-x-xs: Γ⊢⟨Seq x (sequence Seq xs),Normal s ⟩ ⇒ s'
  have Γ⊢sequence Seq ys ↓ s'
  proof −
    from exec-x-xs obtain t where
      exec-x: Γ⊢⟨x,Normal s ⟩ ⇒ t and
      exec-xs: Γ⊢⟨sequence Seq xs,t ⟩ ⇒ s'
      by cases
    show ?thesis
    proof (cases t)
      case (Normal t')
      with exec-x termi-xs-ys have Γ⊢sequence Seq (xs@ys) ↓ Normal t'
        by auto
      from Cons.hyps [OF this] exec-xs Normal
      show ?thesis
        by auto
    next
      case (Abrupt t')
      with exec-xs have s'=Abrupt t'
        by (auto dest: Abrupt-end)
```

```
        thus ?thesis by (auto intro: terminates.intros)
      next
        case (Fault f)
        with exec-xs have s'=Fault f
          by (auto dest: Fault-end)
        thus ?thesis by (auto intro: terminates.intros)
      next
        case Stuck
        with exec-xs have s'=Stuck
          by (auto dest: Stuck-end)
        thus ?thesis by (auto intro: terminates.intros)
      qed
    qed
  }
  ultimately show ?thesis
    using Cons
    by auto
  qed
qed

lemma terminates-sequence-appE [consumes 1]:
  ⟦Γ⊢sequence Seq (xs @ ys) ↓ Normal s;
   ⟦Γ⊢sequence Seq xs ↓ Normal s;
    ∀ s'. Γ⊢⟨sequence Seq xs,Normal s ⟩ ⇒ s' ⟶ Γ⊢sequence Seq ys ↓ s'⟧ ⟹ P⟧
  ⟹ P
  by (auto dest: terminates-sequence-appD)

lemma terminates-to-terminates-sequence-flatten:
  assumes termi: Γ⊢c↓s
  shows Γ⊢sequence Seq (flatten c)↓s
using termi
by (induct)
   (auto intro: terminates.intros terminates-sequence-app
     exec-sequence-flatten-to-exec)

lemma terminates-to-terminates-normalize:
  assumes termi: Γ⊢c↓s
  shows Γ⊢normalize c↓s
using termi
proof induct
  case Seq
  thus ?case
    by (fastforce intro: terminates.intros terminates-sequence-app
                  terminates-to-terminates-sequence-flatten
        dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
next
  case WhileTrue
  thus ?case
    by (fastforce intro: terminates.intros terminates-sequence-app
```

```
             terminates-to-terminates-sequence-flatten
       dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
next
  case Catch
  thus ?case
    by (fastforce intro: terminates.intros terminates-sequence-app
             terminates-to-terminates-sequence-flatten
       dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
qed (auto intro: terminates.intros)

lemma terminates-sequence-flatten-to-terminates:
  shows ⋀s. Γ⊢sequence Seq (flatten c)↓s ⟹ Γ⊢c↓s
proof (induct c)
  case (Seq c1 c2)
  have Γ⊢sequence Seq (flatten (Seq c1 c2)) ↓ s by fact
  hence termi-app: Γ⊢sequence Seq (flatten c1 @ flatten c2) ↓ s by simp
  show ?case
  proof (cases s)
    case (Normal s')
    have Γ⊢Seq c1 c2 ↓ Normal s'
    proof (rule terminates.Seq [rule-format])
      from termi-app [simplified Normal]
      have Γ⊢sequence Seq (flatten c1) ↓ Normal s'
        by (cases rule: terminates-sequence-appE)
      with Seq.hyps
      show Γ⊢c1 ↓ Normal s'
        by simp
    next
      fix s''
      assume Γ⊢⟨c1,Normal s' ⟩ ⇒ s''
      from termi-app [simplified Normal] exec-to-exec-sequence-flatten [OF this]
      have Γ⊢sequence Seq (flatten c2) ↓ s''
        by (cases rule: terminates-sequence-appE) auto
      with Seq.hyps
      show Γ⊢c2 ↓ s''
        by simp
    qed
    with Normal show ?thesis
      by simp
  qed (auto intro: terminates.intros)
qed (auto intro: terminates.intros)

lemma terminates-normalize-to-terminates:
  shows ⋀s. Γ⊢normalize c↓s ⟹ Γ⊢c↓s
proof (induct c)
  case Skip thus ?case by (auto intro: terminates-Skip')
next
  case Basic thus ?case by (cases s) (auto intro: terminates.intros)
next
```

case *Spec* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
case (*Seq c1 c2*)
have Γ⊢*normalize* (*Seq c1 c2*) ↓ *s* **by** *fact*
hence *termi-app*: Γ⊢*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*)) ↓ *s*
by *simp*
show *?case*
proof (*cases s*)
case (*Normal s'*)
have Γ⊢*Seq c1 c2* ↓ *Normal s'*
proof (*rule terminates.Seq* [*rule-format*])
from *termi-app* [*simplified Normal*]
have Γ⊢*sequence Seq* (*flatten* (*normalize c1*)) ↓ *Normal s'*
by (*cases rule*: *terminates-sequence-appE*)
from *terminates-sequence-flatten-to-terminates* [*OF this*] *Seq.hyps*
show Γ⊢*c1* ↓ *Normal s'*
by *simp*
next
fix *s''*
assume Γ⊢⟨*c1*,*Normal s'* ⟩ ⇒ *s''*
from *exec-to-exec-normalize* [*OF this*]
have Γ⊢⟨*normalize c1*,*Normal s'* ⟩ ⇒ *s''* **.**
from *termi-app* [*simplified Normal*] *exec-to-exec-sequence-flatten* [*OF this*]
have Γ⊢*sequence Seq* (*flatten* (*normalize c2*)) ↓ *s''*
by (*cases rule*: *terminates-sequence-appE*) *auto*
from *terminates-sequence-flatten-to-terminates* [*OF this*] *Seq.hyps*
show Γ⊢*c2* ↓ *s''*
by *simp*
qed
with *Normal* **show** *?thesis* **by** *simp*
qed (*auto intro*: *terminates.intros*)
next
case (*Cond b c1 c2*)
thus *?case*
by (*cases s*)
(*auto intro*: *terminates.intros elim!*: *terminates-Normal-elim-cases*)
next
case (*While b c*)
have Γ⊢*normalize* (*While b c*) ↓ *s* **by** *fact*
hence *termi-norm-w*: Γ⊢ *While b* (*normalize c*) ↓ *s* **by** *simp*
{
fix *t w*
assume *termi-w*: Γ⊢ *w* ↓ *t*
have *w*=*While b* (*normalize c*) ⟹ Γ⊢ *While b c* ↓ *t*
using *termi-w*
proof (*induct*)
case (*WhileTrue t' b' c'*)
from *WhileTrue* **obtain**

t′-b: $t' \in b$ **and**
termi-norm-c: $\Gamma\vdash$*normalize c* $\downarrow$ *Normal* t' **and**
termi-norm-w′: $\forall s'$. $\Gamma\vdash\langle$*normalize c*,*Normal* t' $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash$ *While b c* $\downarrow s'$
by *auto*
from *While.hyps* [*OF termi-norm-c*]
have $\Gamma\vdash c \downarrow$ *Normal* t'**.**
moreover
from *termi-norm-w′*
have $\forall s'$. $\Gamma\vdash\langle c$,*Normal* t' $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash$ *While b c* $\downarrow s'$
by (*auto intro*: *exec-to-exec-normalize*)
ultimately show *?case*
using *t′-b*
by (*auto intro*: *terminates.intros*)
qed (*auto intro*: *terminates.intros*)
}
from *this* [*OF termi-norm-w*]
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** *simp*
next
case *DynCom* **thus** *?case*
by (*cases s*) (*auto intro*: *terminates.intros rangeI elim*: *terminates-Normal-elim-cases*)
next
case *Guard* **thus** *?case*
by (*cases s*) (*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *Throw* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
case *Catch*
thus *?case*
by (*cases s*)
(*auto dest*: *exec-to-exec-normalize elim*!: *terminates-Normal-elim-cases*
intro!: *terminates.Catch*)
qed

lemma *terminates-iff-terminates-normalize*:
$\Gamma\vdash$*normalize c*$\downarrow$*s* = $\Gamma\vdash c\downarrow s$
by (*auto intro*: *terminates-to-terminates-normalize*
terminates-normalize-to-terminates)

3.3 Lemmas about *strip-guards*

lemma *terminates-strip-guards-to-terminates*: $\bigwedge s$. $\Gamma\vdash$*strip-guards F c*$\downarrow$*s* $\Longrightarrow \Gamma\vdash c\downarrow s$
proof (*induct c*)
case *Skip* **thus** *?case* **by** *simp*
next
case *Basic* **thus** *?case* **by** *simp*
next

```
  case Spec thus ?case by simp
next
  case (Seq c1 c2)
  hence Γ⊢Seq (strip-guards F c1) (strip-guards F c2) ↓ s by simp
  thus Γ⊢Seq c1 c2 ↓ s
  proof (cases)
    fix f assume s=Fault f thus ?thesis by simp
  next
    assume s=Stuck thus ?thesis by simp
  next
    fix s' assume s=Abrupt s' thus ?thesis by simp
  next
    fix s'
    assume s: s=Normal s'
    assume Γ⊢strip-guards F c1 ↓ Normal s'
    hence Γ⊢c1 ↓ Normal s'
      by (rule Seq.hyps)
    moreover
    assume c2:
      ∀ s''. Γ⊢⟨strip-guards F c1,Normal s'⟩ ⇒ s'' ⟶ Γ⊢strip-guards F c2↓s''
    {
      fix s'' assume exec-c1: Γ⊢⟨c1,Normal s' ⟩ ⇒ s''
      have  Γ⊢c2 ↓ s''
      proof (cases s'')
        case (Normal s''')
        with exec-c1
        have Γ⊢⟨strip-guards F c1,Normal s' ⟩ ⇒ s''
          by (auto intro: exec-to-exec-strip-guards)
        with c2
        show ?thesis
          by (iprover intro: Seq.hyps)
      next
        case (Abrupt s''')
        with exec-c1
        have Γ⊢⟨strip-guards F c1,Normal s' ⟩ ⇒ s''
          by (auto intro: exec-to-exec-strip-guards )
        with c2
        show ?thesis
          by (iprover intro: Seq.hyps)
      next
        case Fault thus ?thesis by simp
      next
        case Stuck thus ?thesis by simp
      qed
    }
    ultimately show ?thesis
      using s
      by (iprover intro: terminates.intros)
  qed
```

```
next
  case (Cond b c1 c2)
  hence Γ⊢Cond b (strip-guards F c1) (strip-guards F c2) ↓ s by simp
  thus Γ⊢Cond b c1 c2 ↓ s
  proof (cases)
    fix f assume s=Fault f thus ?thesis by simp
  next
    assume s=Stuck thus ?thesis by simp
  next
    fix s' assume s=Abrupt s' thus ?thesis by simp
  next
    fix s'
    assume s'∈b Γ⊢strip-guards F c1 ↓ Normal s' s = Normal s'
    thus ?thesis
      by (iprover intro: terminates.intros Cond.hyps)
  next
    fix s'
    assume s'∉b Γ⊢strip-guards F c2 ↓ Normal s' s = Normal s'
    thus ?thesis
      by (iprover intro: terminates.intros Cond.hyps)
  qed
next
  case (While b c)
  have hyp-c: ⋀s. Γ⊢strip-guards F c ↓ s ⟹ Γ⊢c ↓ s by fact
  have Γ⊢While b (strip-guards F c) ↓ s using While.prems by simp
  moreover
  {
    fix sw
    assume Γ⊢sw↓s
    then have sw=While b (strip-guards F c) ⟹
      Γ⊢While b c ↓ s
    proof (induct)
      case (WhileTrue s b' c')
      have eqs: While b' c' = While b (strip-guards F c) by fact
      with ‹s∈b'› have b: s∈b by simp
      from eqs ‹Γ⊢c' ↓ Normal s› have Γ⊢strip-guards F c ↓ Normal s
        by simp
      hence term-c: Γ⊢c ↓ Normal s
        by (rule hyp-c)
      moreover
      {
        fix t
        assume exec-c: Γ⊢⟨c,Normal s ⟩ ⇒ t
        have Γ⊢While b c ↓ t
        proof (cases t)
          case Fault
          thus ?thesis by simp
        next
          case Stuck
```

```
          thus ?thesis by simp
        next
          case (Abrupt t')
          thus ?thesis by simp
        next
          case (Normal t')
          with exec-c
          have Γ⊢⟨strip-guards F c,Normal s ⟩ ⇒ Normal t'
            by (auto intro: exec-to-exec-strip-guards)
          with WhileTrue.hyps eqs Normal
          show ?thesis
            by fastforce
        qed
      }
      ultimately
      show ?case
        using b
        by (auto intro: terminates.intros)
    next
      case WhileFalse thus ?case by (auto intro: terminates.intros)
    qed simp-all
  }
  ultimately show Γ⊢While b c ↓ s
    by auto
next
  case Call thus ?case by simp
next
  case DynCom thus ?case
    by (cases s) (auto elim: terminates-Normal-elim-cases intro: terminates.intros
rangeI)
next
  case Guard
  thus ?case
    by (cases s) (auto elim: terminates-Normal-elim-cases intro: terminates.intros
              split: if-split-asm)
next
  case Throw thus ?case by simp
next
  case (Catch c1 c2)
  hence Γ⊢Catch (strip-guards F c1) (strip-guards F c2) ↓ s by simp
  thus Γ⊢Catch c1 c2 ↓ s
  proof (cases)
    fix f assume s=Fault f thus ?thesis by simp
  next
    assume s=Stuck thus ?thesis by simp
  next
    fix s' assume s=Abrupt s' thus ?thesis by simp
  next
    fix s'
```

```
    assume s: s=Normal s'
    assume Γ⊢strip-guards F c1 ↓ Normal s'
    hence Γ⊢c1 ↓ Normal s'
      by (rule Catch.hyps)
    moreover
    assume c2:
      ∀ s''. Γ⊢⟨strip-guards F c1,Normal s'⟩ ⇒ Abrupt s''
             ⟶ Γ⊢strip-guards F c2↓Normal s''
    {
      fix s'' assume exec-c1: Γ⊢⟨c1,Normal s' ⟩ ⇒ Abrupt s''
      have  Γ⊢c2 ↓ Normal s''
      proof −
        from exec-c1
        have Γ⊢⟨strip-guards F c1,Normal s' ⟩ ⇒ Abrupt s''
          by (auto intro: exec-to-exec-strip-guards)
        with c2
        show ?thesis
          by (auto intro: Catch.hyps)
      qed
    }
    ultimately show ?thesis
      using s
      by (iprover intro: terminates.intros)
  qed
qed

lemma terminates-strip-to-terminates:
  assumes termi-strip: strip F Γ⊢c↓s
  shows Γ⊢c↓s
using termi-strip
proof induct
  case (Seq c1 s c2)
  have Γ⊢c1 ↓ Normal s by fact
  moreover
  {
    fix s'
    assume exec: Γ⊢ ⟨c1,Normal s⟩ ⇒ s'
    have Γ⊢c2 ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis
        by (auto elim: isFaultE)
    next
      case False
      from exec-to-exec-strip [OF exec this] Seq.hyps
      show ?thesis
        by auto
    qed
  }
```

ultimately show *?case*
  by (*auto intro*: *terminates.intros*)
next
  case (*WhileTrue s b c*)
  have $\Gamma \vdash c \downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix s'
    assume *exec*: $\Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow s'$
    have $\Gamma \vdash$ *While b c* $\downarrow s'$
    proof (*cases isFault* s')
      case *True*
      thus *?thesis*
        by (*auto elim*: *isFaultE*)
    next
      case *False*
      from *exec-to-exec-strip* [*OF exec this*] *WhileTrue.hyps*
      show *?thesis*
        by *auto*
    qed
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have $\Gamma \vdash c1 \downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix s'
    assume *exec*: $\Gamma \vdash \langle c1, Normal\ s\rangle \Rightarrow Abrupt\ s'$
    from *exec-to-exec-strip* [*OF exec*] *Catch.hyps*
    have $\Gamma \vdash c2 \downarrow Normal\ s'$
      by *auto*
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
next
  case *Call* **thus** *?case*
    by (*auto intro*: *terminates.intros terminates-strip-guards-to-terminates*)
qed (*auto intro*: *terminates.intros*)

3.4 Lemmas about $c_1 \cap_g c_2$

lemma *inter-guards-terminates*:
  $\bigwedge c\ c2\ s.\ [\![(c1 \cap_g c2) = Some\ c; \Gamma \vdash c1 \downarrow s\ ]\!]$
      $\Longrightarrow \Gamma \vdash c \downarrow s$
proof (*induct c1*)
  case *Skip* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Skip*)
next

```
  case (Basic f) thus ?case by (fastforce simp add: inter-guards-Basic)
next
  case (Spec r) thus ?case by (fastforce simp add: inter-guards-Spec)
next
  case (Seq a1 a2)
  have (Seq a1 a2 ∩g c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Seq b1 b2 and
    d1: (a1 ∩g b1) = Some d1 and d2: (a2 ∩g b2) = Some d2 and
    c: c=Seq d1 d2
    by (auto simp add: inter-guards-Seq)
  have termi-c1: Γ⊢Seq a1 a2 ↓ s by fact
  have Γ⊢Seq d1 d2 ↓ s
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    note Normal-s = this
    with d1 termi-c1
    have Γ⊢d1 ↓ Normal s'
      by (auto elim: terminates-Normal-elim-cases intro: Seq.hyps)
    moreover
    {
      fix t
      assume exec-d1: Γ⊢⟨d1,Normal s' ⟩ ⇒ t
      have Γ⊢d2 ↓ t
      proof (cases t)
        case Fault thus ?thesis by simp
      next
        case Stuck thus ?thesis by simp
      next
        case Abrupt thus ?thesis by simp
      next
        case (Normal t')
        with inter-guards-exec-noFault [OF d1 exec-d1]
        have Γ⊢⟨a1,Normal s' ⟩ ⇒ Normal t'
          by simp
        with termi-c1 Normal-s have Γ⊢a2 ↓ Normal t'
          by (auto elim: terminates-Normal-elim-cases)
        with d2 have Γ⊢d2 ↓ Normal t'
          by (auto intro: Seq.hyps)
        with Normal show ?thesis by simp
      qed
    }
    ultimately have Γ⊢Seq d1 d2 ↓ Normal s'
```

```
      by (fastforce intro: terminates.intros)
    with Normal show ?thesis by simp
  qed
  with c show ?case by simp
next
  case Cond thus ?case
    by − (cases s,
          auto intro: terminates.intros elim!: terminates-Normal-elim-cases
               simp add: inter-guards-Cond)
next
  case (While b bdy1)
  have (While b bdy1 ∩g c2) = Some c by fact
  then obtain bdy2 bdy where
    c2: c2=While b bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=While b bdy
    by (auto simp add: inter-guards-While)
  have Γ⊢While b bdy1 ↓ s by fact
  moreover
  {
    fix s w w1 w2
    assume termi-w:  Γ⊢w ↓ s
    assume w: w=While b bdy1
    from termi-w w
    have Γ⊢While b bdy ↓ s
    proof (induct)
      case (WhileTrue s b' bdy1')
      have eqs: While b' bdy1' = While b bdy1 by fact
      from WhileTrue have s-in-b: s ∈ b by simp
      from WhileTrue have termi-bdy1: Γ⊢bdy1 ↓ Normal s by simp
      show ?case
      proof −
        from bdy termi-bdy1
        have Γ⊢bdy↓(Normal s)
          by (rule While.hyps)
        moreover
        {
          fix t
          assume exec-bdy: Γ⊢⟨bdy,Normal s ⟩ ⇒ t
          have Γ⊢While b bdy↓t
          proof (cases t)
            case Fault thus ?thesis by simp
          next
            case Stuck thus ?thesis by simp
          next
            case Abrupt thus ?thesis by simp
          next
            case (Normal t')
            with inter-guards-exec-noFault [OF bdy exec-bdy]
```

```
          have Γ⊢⟨bdy1,Normal s ⟩ ⇒ Normal t′
            by simp
          with WhileTrue have Γ⊢ While b bdy ↓ Normal t′
            by simp
          with Normal show ?thesis by simp
        qed
      }
      ultimately show ?thesis
        using s-in-b
        by (blast intro: terminates.WhileTrue)
    qed
  next
    case WhileFalse thus ?case
      by (blast intro: terminates.WhileFalse)
  qed (simp-all)
}
ultimately
show ?case using c by simp
next
  case Call thus ?case by (simp add: inter-guards-Call)
next
  case (DynCom f1)
  have (DynCom f1 ∩g c2) = Some c by fact
  then obtain f2 f where
    c2: c2=DynCom f2 and
    f-defined: ∀ s. ((f1 s) ∩g (f2 s)) ≠ None and
    c: c=DynCom (λs. the ((f1 s) ∩g (f2 s)))
    by (auto simp add: inter-guards-DynCom)
  have termi: Γ⊢DynCom f1 ↓ s by fact
  show ?case
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s′)
    from f-defined obtain f where f: ((f1 s′) ∩g (f2 s′)) = Some f
      by auto
    from Normal termi
    have Γ⊢f1 s′↓ (Normal s′)
      by (auto elim: terminates-Normal-elim-cases)
    from DynCom.hyps f this
    have Γ⊢f↓ (Normal s′)
      by blast
    with c f Normal
    show ?thesis
      by (auto intro: terminates.intros)
```

```
  qed
next
  case (Guard f g1 bdy1)
  have (Guard f g1 bdy1 ∩g c2) = Some c by fact
  then obtain g2 bdy2 bdy where
    c2: c2=Guard f g2 bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=Guard f (g1 ∩ g2) bdy
    by (auto simp add: inter-guards-Guard)
  have termi-c1: Γ⊢Guard f g1 bdy1 ↓ s by fact
  show ?case
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    show ?thesis
    proof (cases s' ∈ g1)
      case False
      with Normal c show ?thesis by (auto intro: terminates.GuardFault)
    next
      case True
      note s-in-g1 = this
      show ?thesis
      proof (cases s' ∈ g2)
        case False
        with Normal c show ?thesis by (auto intro: terminates.GuardFault)
      next
        case True
        with termi-c1 s-in-g1 Normal have Γ⊢bdy1 ↓ Normal s'
          by (auto elim: terminates-Normal-elim-cases)
        with c bdy Guard.hyps Normal True s-in-g1
        show ?thesis by (auto intro: terminates.Guard)
      qed
    qed
  qed
next
  case Throw thus ?case
    by (auto simp add: inter-guards-Throw)
next
  case (Catch a1 a2)
  have (Catch a1 a2 ∩g c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Catch b1 b2 and
    d1: (a1 ∩g b1) = Some d1 and d2: (a2 ∩g b2) = Some d2 and
    c: c=Catch d1 d2
```

```
    by (auto simp add: inter-guards-Catch)
  have termi-c1: Γ⊢Catch a1 a2 ↓ s by fact
  have Γ⊢Catch d1 d2 ↓ s
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    note Normal-s = this
    with d1 termi-c1
    have Γ⊢d1 ↓ Normal s'
      by (auto elim: terminates-Normal-elim-cases intro: Catch.hyps)
    moreover
    {
      fix t
      assume exec-d1: Γ⊢⟨d1,Normal s' ⟩ ⇒ Abrupt t
      have Γ⊢d2 ↓ Normal t
      proof −
        from inter-guards-exec-noFault [OF d1 exec-d1]
        have Γ⊢⟨a1,Normal s' ⟩ ⇒ Abrupt t
          by simp
        with termi-c1 Normal-s have Γ⊢a2 ↓ Normal t
          by (auto elim: terminates-Normal-elim-cases)
        with d2 have Γ⊢d2 ↓ Normal t
          by (auto intro: Catch.hyps)
        with Normal show ?thesis by simp
      qed
    }
    ultimately have Γ⊢Catch d1 d2 ↓ Normal s'
      by (fastforce intro: terminates.intros)
    with Normal show ?thesis by simp
  qed
  with c show ?case by simp
qed

lemma inter-guards-terminates':
  assumes c: (c1 ∩g c2) = Some c
  assumes termi-c2: Γ⊢c2↓s
  shows Γ⊢c↓s
proof −
  from c have (c2 ∩g c1) = Some c
    by (rule inter-guards-sym)
  from this termi-c2 show ?thesis
    by (rule inter-guards-terminates)
qed
```

3.5 Lemmas about *mark-guards*

lemma *terminates-to-terminates-mark-guards*:
  assumes *termi*: $\Gamma\vdash c\downarrow s$
  shows $\Gamma\vdash$*mark-guards f c*$\downarrow s$
using *termi*
proof (*induct*)
  case *Skip* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Guard* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Fault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Seq c1 s c2*)
  have $\Gamma\vdash$*mark-guards f c1* $\downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix *t*
    assume *exec-mark*: $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow t$
    have $\Gamma\vdash$*mark-guards f c2* $\downarrow t$
    proof −
      from *exec-mark-guards-to-exec* [*OF exec-mark*] **obtain** t' **where**
        exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow t'$ **and**
        t-Fault: *isFault t* $\longrightarrow$ *isFault* t' **and**
        t'*-Fault-f*: t' = *Fault f* $\longrightarrow t' = t$ **and**
        t'*-Fault*: *isFault* t' $\longrightarrow$ *isFault t* **and**
        t'*-noFault*: ¬ *isFault* t' $\longrightarrow t' = t$
        by *blast*
      show *?thesis*
      proof (*cases isFault* t')
        case *True*
        with t'*-Fault* **have** *isFault t* **by** *simp*
        thus *?thesis*
          by (*auto elim*: *isFaultE*)
      next
        case *False*
        with t'*-noFault* **have** t'=t **by** *simp*
        with *exec-c1 Seq.hyps*
        show *?thesis*
          by *auto*
      qed
    qed
  }
  ultimately show *?case*

by (*auto intro*: *terminates.intros*)
next
case *CondTrue* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *CondFalse* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case (*WhileTrue s b c*)
have *s-in-b*: $s \in b$ **by** *fact*
have $\Gamma \vdash$*mark-guards f c* $\downarrow$ *Normal s* **by** *fact*
moreover
{
fix *t*
assume *exec-mark*: $\Gamma \vdash \langle$*mark-guards f c*,*Normal s* $\rangle \Rightarrow t$
have $\Gamma \vdash$*mark-guards f* (*While b c*) $\downarrow t$
proof −
from *exec-mark-guards-to-exec* [*OF exec-mark*] **obtain** t' **where**
exec-c1: $\Gamma \vdash \langle c$,*Normal s* $\rangle \Rightarrow t'$ **and**
t-Fault: *isFault t* $\longrightarrow$ *isFault* t' **and**
t'*-Fault-f*: $t' =$ *Fault f* $\longrightarrow t' = t$ **and**
t'*-Fault*: *isFault* t' $\longrightarrow$ *isFault t* **and**
t'*-noFault*: $\neg$ *isFault* t' $\longrightarrow t' = t$
by *blast*
show *?thesis*
proof (*cases isFault* t')
case *True*
with t'*-Fault* **have** *isFault t* **by** *simp*
thus *?thesis*
by (*auto elim*: *isFaultE*)
next
case *False*
with t'*-noFault* **have** t'=t **by** *simp*
with *exec-c1 WhileTrue.hyps*
show *?thesis*
by *auto*
qed
qed
}
ultimately show *?case*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Call* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Stuck* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)

next
  case *Throw* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Abrupt* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have $\Gamma\vdash$*mark-guards f c1* $\downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix *t*
    assume *exec-mark*: $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt t*
    have $\Gamma\vdash$*mark-guards f c2* $\downarrow$ *Normal t*
    proof −
      from *exec-mark-guards-to-exec* [*OF exec-mark*] **obtain** t' **where**
        exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow t'$ **and**
        t'*-Fault-f*: t' = *Fault f* $\longrightarrow$ t' = *Abrupt t* **and**
        t'*-Fault*: *isFault* t' $\longrightarrow$ *isFault* (*Abrupt t*) **and**
        t'*-noFault*: $\neg$ *isFault* t' $\longrightarrow$ t' = *Abrupt t*
        by *fastforce*
      show *?thesis*
      proof (*cases isFault* t')
        case *True*
        with t'*-Fault* **have** *isFault* (*Abrupt t*) **by** *simp*
        thus *?thesis* **by** *simp*
      next
        case *False*
        with t'*-noFault* **have** t'=*Abrupt t* **by** *simp*
        with *exec-c1 Catch.hyps*
        show *?thesis*
          by *auto*
      qed
    qed
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
qed

lemma *terminates-mark-guards-to-terminates-Normal*:
  $\bigwedge s.$ $\Gamma\vdash$*mark-guards f c*$\downarrow$*Normal s* $\Longrightarrow$ $\Gamma\vdash c\downarrow$*Normal s*
proof (*induct c*)
  case *Skip* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Seq c1 c2*)
  have $\Gamma\vdash$*mark-guards f* (*Seq c1 c2*) $\downarrow$ *Normal s* **by** *fact*
  then obtain

```
    termi-merge-c1: Γ⊢mark-guards f c1 ↓ Normal s and
    termi-merge-c2: ∀ s'. Γ⊢⟨mark-guards f c1,Normal s ⟩ ⇒ s' ⟶
                        Γ⊢mark-guards f c2 ↓ s'
    by (auto elim: terminates-Normal-elim-cases)
  from termi-merge-c1 Seq.hyps
  have Γ⊢c1 ↓ Normal s by iprover
  moreover
  {
    fix s'
    assume exec-c1: Γ⊢⟨c1,Normal s ⟩ ⇒ s'
    have Γ⊢ c2 ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE)
    next
      case False
      from exec-to-exec-mark-guards [OF exec-c1 False]
      have Γ⊢⟨mark-guards f c1,Normal s ⟩ ⇒ s' .
      from termi-merge-c2 [rule-format, OF this] Seq.hyps
      show ?thesis
        by (cases s') (auto)
    qed
  }
  ultimately show ?case by (auto intro: terminates.intros)
next
  case Cond thus ?case
    by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (While b c)
  {
    fix u c'
    assume termi-c': Γ⊢c' ↓ Normal u
    assume c': c' = mark-guards f (While b c)
    have Γ⊢While b c ↓ Normal u
      using termi-c' c'
    proof (induct)
      case (WhileTrue s b' c')
      have s-in-b: s ∈ b using WhileTrue by simp
      have Γ⊢mark-guards f c ↓ Normal s
        using WhileTrue by (auto elim: terminates-Normal-elim-cases)
      with While.hyps have Γ⊢c ↓ Normal s
        by auto
      moreover
      have hyp-w: ∀ w. Γ⊢⟨mark-guards f c,Normal s ⟩ ⇒ w ⟶ Γ⊢While b c ↓ w
        using WhileTrue by simp
      hence ∀ w. Γ⊢⟨c,Normal s ⟩ ⇒ w ⟶ Γ⊢While b c ↓ w
        apply −
        apply (rule allI)
        apply (case-tac w)
```

```
      apply (auto dest: exec-to-exec-mark-guards)
      done
    ultimately show ?case
      using s-in-b
      by (auto intro: terminates.intros)
  next
    case WhileFalse thus ?case by (auto intro: terminates.intros)
  qed auto
 }
 with While show ?case by simp
next
 case Call thus ?case
  by (fastforce intro: terminates.intros )
next
 case DynCom thus ?case
  by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
 case (Guard f g c)
 thus ?case by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
 case Throw thus ?case
  by (fastforce intro: terminates.intros )
next
 case (Catch c1 c2)
 have Γ⊢mark-guards f (Catch c1 c2) ↓ Normal s by fact
 then obtain
  termi-merge-c1: Γ⊢mark-guards f c1 ↓ Normal s and
  termi-merge-c2: ∀ s'. Γ⊢⟨mark-guards f c1,Normal s ⟩ ⇒ Abrupt s' ⟶
                   Γ⊢mark-guards f c2 ↓ Normal s'
  by (auto elim: terminates-Normal-elim-cases)
 from termi-merge-c1 Catch.hyps
 have Γ⊢c1 ↓ Normal s by iprover
 moreover
 {
  fix s'
  assume exec-c1: Γ⊢⟨c1,Normal s ⟩ ⇒ Abrupt s'
  have Γ⊢ c2 ↓ Normal s'
  proof -
    from exec-to-exec-mark-guards [OF exec-c1]
    have Γ⊢⟨mark-guards f c1,Normal s ⟩ ⇒ Abrupt s' by simp
    from termi-merge-c2 [rule-format, OF this] Catch.hyps
    show ?thesis
      by iprover
  qed
 }
 ultimately show ?case by (auto intro: terminates.intros)
qed

lemma terminates-mark-guards-to-terminates:
```

$\Gamma\vdash$*mark-guards f c*$\downarrow$*s* $\Longrightarrow$ $\Gamma\vdash$*c*$\downarrow$ *s*
by (*cases s*) (*auto intro*: *terminates-mark-guards-to-terminates-Normal*)

3.6 Lemmas about *merge-guards*

lemma *terminates-to-terminates-merge-guards*:
  assumes *termi*: $\Gamma\vdash c\downarrow s$
  shows $\Gamma\vdash$*merge-guards c*$\downarrow$*s*
using *termi*
proof (*induct*)
  case (*Guard s g c f*)
  have *s-in-g*: $s \in g$ **by** *fact*
  have *termi-merge-c*: $\Gamma\vdash$*merge-guards c* $\downarrow$ *Normal s* **by** *fact*
  show *?case*
  proof (*cases* $\exists f'\ g'\ c'$. *merge-guards c* = *Guard* $f'\ g'\ c'$)
    case *False*
    hence *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
      by (*cases merge-guards c*) (*auto simp add*: *Let-def*)
    with *s-in-g termi-merge-c* **show** *?thesis*
      by (*auto intro*: *terminates.intros*)
  next
    case *True*
    then obtain $f'\ g'\ c'$ **where**
      mc: *merge-guards c* = *Guard* $f'\ g'\ c'$
      by *blast*
    show *?thesis*
    proof (*cases* $f=f'$)
      case *False*
      with *mc* **have** *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
        by (*simp add*: *Let-def*)
      with *s-in-g termi-merge-c* **show** *?thesis*
        by (*auto intro*: *terminates.intros*)
    next
      case *True*
      with *mc* **have** *merge-guards* (*Guard f g c*) = *Guard f* $(g \cap g')\ c'$
        by *simp*
      with *s-in-g mc True termi-merge-c*
      show *?thesis*
        by (*cases* $s \in g'$)
          (*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
    qed
  qed
next
  case (*GuardFault s g f c*)
  have $s \notin g$ **by** *fact*
  thus *?case*
    by (*cases merge-guards c*)
      (*auto intro*: *terminates.intros split*: *if-split-asm simp add*: *Let-def*)
qed (*fastforce intro*: *terminates.intros dest*: *exec-merge-guards-to-exec*)+

lemma *terminates-merge-guards-to-terminates-Normal*:
  shows $\bigwedge s.\ \Gamma\vdash$*merge-guards c*$\downarrow$*Normal s* $\Longrightarrow$ $\Gamma\vdash c\downarrow$*Normal s*
proof (*induct c*)
  case *Skip* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Seq c1 c2*)
  have $\Gamma\vdash$*merge-guards* (*Seq c1 c2*) $\downarrow$ *Normal s* **by** *fact*
  then obtain
    termi-merge-c1: $\Gamma\vdash$*merge-guards c1* $\downarrow$ *Normal s* **and**
    termi-merge-c2: $\forall s'.\ \Gamma\vdash\langle$*merge-guards c1*,*Normal s* $\rangle \Rightarrow s' \longrightarrow$
                        $\Gamma\vdash$*merge-guards c2* $\downarrow s'$
    by (*auto elim*: *terminates-Normal-elim-cases*)
  from *termi-merge-c1 Seq.hyps*
  have $\Gamma\vdash$*c1* $\downarrow$ *Normal s* **by** *iprover*
  moreover
  {
    fix s'
    assume *exec-c1*: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow s'$
    have $\Gamma\vdash$ *c2* $\downarrow s'$
    proof −
      from *exec-to-exec-merge-guards* [*OF exec-c1*]
      have $\Gamma\vdash\langle$*merge-guards c1*,*Normal s* $\rangle \Rightarrow s'$ **.**
      from *termi-merge-c2* [*rule-format*, *OF this*] *Seq.hyps*
      show *?thesis*
        by (*cases* s') (*auto*)
    qed
  }
  ultimately show *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Cond* **thus** *?case*
    by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
  case (*While b c*)
  {
    fix *u* c'
    assume *termi-c'*: $\Gamma\vdash c'\downarrow$ *Normal u*
    assume c': c' = *merge-guards* (*While b c*)
    have $\Gamma\vdash$ *While b c* $\downarrow$ *Normal u*
      using *termi-c'* c'
    proof (*induct*)
      case (*WhileTrue s b' c'*)
      have *s-in-b*: $s \in b$ **using** *WhileTrue* **by** *simp*
      have $\Gamma\vdash$*merge-guards c* $\downarrow$ *Normal s*
        using *WhileTrue* **by** (*auto elim*: *terminates-Normal-elim-cases*)

```
      with While.hyps have Γ⊢c ↓ Normal s
        by auto
      moreover
      have hyp-w: ∀ w. Γ⊢⟨merge-guards c,Normal s ⟩ ⇒ w ⟶ Γ⊢While b c ↓ w
        using WhileTrue by simp
      hence ∀ w. Γ⊢⟨c,Normal s ⟩ ⇒ w ⟶ Γ⊢While b c ↓ w
        by (simp add: exec-iff-exec-merge-guards [symmetric])
      ultimately show ?case
        using s-in-b
        by (auto intro: terminates.intros)
    next
      case WhileFalse thus ?case by (auto intro: terminates.intros)
    qed auto
  }
  with While show ?case by simp
next
  case Call thus ?case
    by (fastforce intro: terminates.intros )
next
  case DynCom thus ?case
    by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (Guard f g c)
  have termi-merge: Γ⊢merge-guards (Guard f g c) ↓ Normal s by fact
  show ?case
  proof (cases ∃f' g' c'. merge-guards c = Guard f' g' c')
    case False
    hence m: merge-guards (Guard f g c) = Guard f g (merge-guards c)
      by (cases merge-guards c) (auto simp add: Let-def)
    from termi-merge Guard.hyps show ?thesis
      by (simp only: m)
         (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
  next
    case True
    then obtain f' g' c' where
      mc: merge-guards c = Guard f' g' c'
      by blast
    show ?thesis
    proof (cases f=f')
      case False
      with mc have m: merge-guards (Guard f g c) = Guard f g (merge-guards c)
        by (simp add: Let-def)
      from termi-merge Guard.hyps show ?thesis
      by (simp only: m)
         (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
    next
      case True
      with mc have m: merge-guards (Guard f g c) = Guard f (g ∩ g') c'
        by simp
```

from *termi-merge Guard.hyps*
show *?thesis*
by (*simp only*: *m mc*)
(*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
qed
qed
next
case *Throw* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case (*Catch c1 c2*)
have $\Gamma\vdash$*merge-guards* (*Catch c1 c2*) $\downarrow$ *Normal s* **by** *fact*
then obtain
termi-merge-c1: $\Gamma\vdash$*merge-guards c1* $\downarrow$ *Normal s* **and**
termi-merge-c2: $\forall s'$. $\Gamma\vdash\langle$*merge-guards c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* $s' \longrightarrow$
$\Gamma\vdash$*merge-guards c2* $\downarrow$ *Normal* s'
by (*auto elim*: *terminates-Normal-elim-cases*)
from *termi-merge-c1 Catch.hyps*
have $\Gamma\vdash$*c1* $\downarrow$ *Normal s* **by** *iprover*
moreover
{
fix s'
assume *exec-c1*: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* s'
have $\Gamma\vdash$ *c2* $\downarrow$ *Normal* s'
proof −
from *exec-to-exec-merge-guards* [*OF exec-c1*]
have $\Gamma\vdash\langle$*merge-guards c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* s' **.**
from *termi-merge-c2* [*rule-format*, *OF this*] *Catch.hyps*
show *?thesis*
by *iprover*
qed
}
ultimately show *?case* **by** (*auto intro*: *terminates.intros*)
qed

lemma *terminates-merge-guards-to-terminates*:
$\Gamma\vdash$*merge-guards c*$\downarrow$ *s* $\Longrightarrow$ $\Gamma\vdash$*c*$\downarrow$ *s*
by (*cases s*) (*auto intro*: *terminates-merge-guards-to-terminates-Normal*)

theorem *terminates-iff-terminates-merge-guards*:
$\Gamma\vdash$*c*$\downarrow$ *s* = $\Gamma\vdash$*merge-guards c*$\downarrow$ *s*
by (*iprover intro*: *terminates-to-terminates-merge-guards*
terminates-merge-guards-to-terminates)

3.7 Lemmas about $c_1 \subseteq_g c_2$

lemma *terminates-fewer-guards-Normal*:
shows $\bigwedge c\ s.$ $[\![\Gamma\vdash c'\downarrow Normal\ s;\ c \subseteq_g c';\ \Gamma\vdash\langle c',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV]\!]$
$\Longrightarrow \Gamma\vdash c\downarrow Normal\ s$

```
proof (induct c')
  case Skip thus ?case by (auto intro: terminates.intros dest: subseteq-guardsD)
next
  case Basic thus ?case by (auto intro: terminates.intros dest: subseteq-guardsD)
next
  case Spec thus ?case by (auto intro: terminates.intros dest: subseteq-guardsD)
next
  case (Seq c1' c2')
  have termi: Γ⊢Seq c1' c2' ↓ Normal s by fact
  then obtain
    termi-c1': Γ⊢c1'↓ Normal s and
    termi-c2': ∀ s'. Γ⊢⟨c1',Normal s ⟩ ⇒ s' ⟶ Γ⊢c2'↓ s'
    by (auto elim: terminates-Normal-elim-cases)
  have noFault: Γ⊢⟨Seq c1' c2',Normal s ⟩ ⇒∉Fault ` UNIV by fact
  hence noFault-c1': Γ⊢⟨c1',Normal s ⟩ ⇒∉Fault ` UNIV
    by (auto intro: exec.intros simp add: final-notin-def)
  have c ⊆g Seq c1' c2' by fact
  from subseteq-guards-Seq [OF this] obtain c1 c2 where
    c: c = Seq c1 c2 and
    c1-c1': c1 ⊆g c1' and
    c2-c2': c2 ⊆g c2'
    by blast
  from termi-c1' c1-c1' noFault-c1'
  have Γ⊢c1↓ Normal s
    by (rule Seq.hyps)
  moreover
  {
    fix t
    assume exec-c1: Γ⊢⟨c1,Normal s ⟩ ⇒ t
    have Γ⊢c2↓ t
    proof -
      from exec-to-exec-subseteq-guards [OF c1-c1' exec-c1] obtain t' where
        exec-c1': Γ⊢⟨c1',Normal s ⟩ ⇒ t' and
        t-Fault: isFault t ⟶ isFault t' and
        t'-noFault: ¬ isFault t' ⟶ t' = t
        by blast
      show ?thesis
      proof (cases isFault t')
        case True
        with exec-c1' noFault-c1'
        have False
          by (fastforce elim: isFaultE dest: Fault-end simp add: final-notin-def)
        thus ?thesis ..
      next
        case False
        with t'-noFault have t': t'=t by simp
        with termi-c2' exec-c1'
        have termi-c2': Γ⊢c2'↓ t
          by auto
```

show *?thesis*
proof (*cases t*)
case *Fault* **thus** *?thesis* **by** *auto*
next
case *Abrupt* **thus** *?thesis* **by** *auto*
next
case *Stuck* **thus** *?thesis* **by** *auto*
next
case (*Normal u*)
with *noFault exec-c1′ t′*
have $\Gamma\vdash\langle c2', Normal\ u\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c2′* [*simplified Normal*] *c2-c2′ this*
have $\Gamma\vdash c2 \downarrow Normal\ u$
by (*rule Seq.hyps*)
with *Normal exec-c1*
show *?thesis* **by** *simp*
qed
qed
qed
}
ultimately show *?case* **using** *c* **by** (*auto intro*: *terminates.intros*)
next
case (*Cond b c1′ c2′*)
have *noFault*: $\Gamma\vdash\langle Cond\ b\ c1'\ c2', Normal\ s\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV* **by** *fact*
have *termi*: $\Gamma\vdash Cond\ b\ c1'\ c2' \downarrow Normal\ s$ **by** *fact*
have $c \subseteq_g Cond\ b\ c1'\ c2'$ **by** *fact*
from *subseteq-guards-Cond* [*OF this*] **obtain** *c1 c2* **where**
c: $c = Cond\ b\ c1\ c2$ **and**
c1-c1′: $c1 \subseteq_g c1'$ **and**
c2-c2′: $c2 \subseteq_g c2'$
by *blast*
thus *?case*
proof (*cases* $s \in b$)
case *True*
with *termi* **have** *termi-c1′*: $\Gamma\vdash c1' \downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *True noFault* **have** $\Gamma\vdash\langle c1', Normal\ s\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c1′ c1-c1′ this*
have $\Gamma\vdash c1 \downarrow Normal\ s$
by (*rule Cond.hyps*)
with *True c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
next
case *False*
with *termi* **have** *termi-c2′*: $\Gamma\vdash c2' \downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *False noFault* **have** $\Gamma\vdash\langle c2', Normal\ s\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*

by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c2′ c2-c2′ this*
have $\Gamma\vdash c2 \downarrow Normal\ s$
by (*rule Cond.hyps*)
with *False c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*While b c′*)
have *noFault*: $\Gamma\vdash\langle While\ b\ c',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
have *termi*: $\Gamma\vdash While\ b\ c' \downarrow Normal\ s$ **by** *fact*
have $c \subseteq_g While\ b\ c'$ **by** *fact*
from *subseteq-guards-While* [*OF this*]
obtain c'' **where**
c: $c = While\ b\ c''$ **and**
c''-c': $c'' \subseteq_g c'$
by *blast*
{
fix *d u*
assume *termi*: $\Gamma\vdash d \downarrow u$
assume *d*: $d = While\ b\ c'$
assume *noFault*: $\Gamma\vdash\langle While\ b\ c',u\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
have $\Gamma\vdash While\ b\ c'' \downarrow u$
using *termi d noFault*
proof (*induct*)
case (*WhileTrue u b′ c‴*)
have *u-in-b*: $u \in b$ **using** *WhileTrue* **by** *simp*
have *termi-c′*: $\Gamma\vdash c' \downarrow Normal\ u$ **using** *WhileTrue* **by** *simp*
have *noFault*: $\Gamma\vdash\langle While\ b\ c',Normal\ u\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **using** *WhileTrue*
by *simp*
hence *noFault-c′*: $\Gamma\vdash\langle c',Normal\ u\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **using** *u-in-b*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *While.hyps* [*OF termi-c′ c″-c′ this*]
have $\Gamma\vdash c'' \downarrow Normal\ u$**.**
moreover
from *WhileTrue*
have *hyp-w*: $\forall s'.\ \Gamma\vdash\langle c',Normal\ u\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash\langle While\ b\ c',s'\ \rangle \Rightarrow\notin Fault\ `$
$UNIV$
$\longrightarrow \Gamma\vdash While\ b\ c'' \downarrow s'$
by *simp*
{
fix *v*
assume *exec-c″*: $\Gamma\vdash\langle c'',Normal\ u\ \rangle \Rightarrow v$
have $\Gamma\vdash While\ b\ c'' \downarrow v$
proof −
from *exec-to-exec-subseteq-guards* [*OF c″-c′ exec-c″*] **obtain** v' **where**
exec-c′: $\Gamma\vdash\langle c',Normal\ u\ \rangle \Rightarrow v'$ **and**
v-Fault: $isFault\ v \longrightarrow isFault\ v'$ **and**
v'-*noFault*: $\neg\ isFault\ v' \longrightarrow v' = v$

```
          by auto
        show ?thesis
        proof (cases isFault v')
          case True
          with exec-c' noFault u-in-b
          have False
            by (fastforce
                  simp add: final-notin-def intro: exec.intros elim: isFaultE)
          thus ?thesis ..
        next
          case False
          with v'-noFault have v': v'=v
            by simp
          with noFault exec-c' u-in-b
          have Γ⊢⟨While b c',v ⟩ ⇒∉Fault ‘ UNIV
            by (fastforce simp add: final-notin-def intro: exec.intros)
          from hyp-w [rule-format, OF exec-c' [simplified v'] this]
          show Γ⊢While b c'' ↓ v .
        qed
      qed
    }
    ultimately
    show ?case using u-in-b
      by (auto intro: terminates.intros)
  next
    case WhileFalse thus ?case by (auto intro: terminates.intros)
  qed auto
}
with c noFault termi show ?case
  by auto
next
case Call thus ?case by (auto intro: terminates.intros dest: subseteq-guardsD)
next
case (DynCom C')
have termi: Γ⊢DynCom C' ↓ Normal s by fact
hence termi-C': Γ⊢C' s ↓ Normal s
  by cases
have noFault: Γ⊢⟨DynCom C',Normal s ⟩ ⇒∉Fault ‘ UNIV by fact
hence noFault-C': Γ⊢⟨C' s,Normal s ⟩ ⇒∉Fault ‘ UNIV
  by (auto intro: exec.intros simp add: final-notin-def)
have c ⊆g DynCom C' by fact
from subseteq-guards-DynCom [OF this] obtain C where
  c: c = DynCom C and
  C-C': ∀ s. C s ⊆g C' s
  by blast
from DynCom.hyps termi-C' C-C' [rule-format] noFault-C'
have Γ⊢C s ↓ Normal s
  by fast
with c show ?case
```

by (*auto intro*: *terminates.intros*)
next
case (*Guard* f' g' c')
have *noFault*: $\Gamma\vdash\langle Guard\ f'\ g'\ c',Normal\ s\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV* **by** *fact*
have *termi*: $\Gamma\vdash Guard\ f'\ g'\ c' \downarrow Normal\ s$ **by** *fact*
have $c \subseteq_g Guard\ f'\ g'\ c'$ **by** *fact*
hence *c-cases*: $(c \subseteq_g c') \vee (\exists c''.\ c = Guard\ f'\ g'\ c'' \wedge (c'' \subseteq_g c'))$
by (*rule subseteq-guards-Guard*)
thus *?case*
proof (*cases* $s \in g'$)
case *True*
note *s-in-g'* = *this*
with *noFault* **have** *noFault-c'*: $\Gamma\vdash\langle c',Normal\ s\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
from *termi s-in-g'* **have** *termi-c'*: $\Gamma\vdash c' \downarrow Normal\ s$
by *cases auto*
from *c-cases* **show** *?thesis*
proof
assume $c \subseteq_g c'$
from *termi-c' this noFault-c'*
show $\Gamma\vdash c \downarrow Normal\ s$
by (*rule Guard.hyps*)
next
assume $\exists c''.\ c = Guard\ f'\ g'\ c'' \wedge (c'' \subseteq_g c')$
then obtain c'' **where**
c: $c = Guard\ f'\ g'\ c''$ **and** *c''-c'*: $c'' \subseteq_g c'$
by *blast*
from *termi-c' c''-c' noFault-c'*
have $\Gamma\vdash c'' \downarrow Normal\ s$
by (*rule Guard.hyps*)
with *s-in-g' c*
show *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case *False*
with *noFault* **have** *False*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
thus *?thesis* **..**
qed
next
case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
case (*Catch c1' c2'*)
have *termi*: $\Gamma\vdash Catch\ c1'\ c2' \downarrow Normal\ s$ **by** *fact*
then obtain
termi-c1': $\Gamma\vdash c1' \downarrow Normal\ s$ **and**
termi-c2': $\forall s'.\ \Gamma\vdash\langle c1',Normal\ s\ \rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash c2' \downarrow Normal\ s'$
by (*auto elim*: *terminates-Normal-elim-cases*)

have *noFault*: $\Gamma\vdash\langle$*Catch c1′ c2′*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *UNIV* **by** *fact*
hence *noFault-c1′*: $\Gamma\vdash\langle$*c1′*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *UNIV*
  by (*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
have $c \subseteq_g$ *Catch c1′ c2′* **by** *fact*
from *subseteq-guards-Catch* [*OF this*] **obtain** *c1 c2* **where**
  c: *c* = *Catch c1 c2* **and**
  c1-c1′: $c1 \subseteq_g c1'$ **and**
  c2-c2′: $c2 \subseteq_g c2'$
  by *blast*
from *termi-c1′ c1-c1′ noFault-c1′*
have $\Gamma\vdash$*c1*$\downarrow$ *Normal s*
  by (*rule Catch.hyps*)
moreover
{
  fix *t*
  assume *exec-c1*: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt t*
  have $\Gamma\vdash$*c2*$\downarrow$ *Normal t*
  proof −
    from *exec-to-exec-subseteq-guards* [*OF c1-c1′ exec-c1*] **obtain** *t′* **where**
      exec-c1′: $\Gamma\vdash\langle$*c1′*,*Normal s* $\rangle \Rightarrow$ *t′* **and**
      t′-noFault: $\neg$ *isFault t′* $\longrightarrow$ *t′* = *Abrupt t*
      by *blast*
    show *?thesis*
    proof (*cases isFault t′*)
      case *True*
      with *exec-c1′ noFault-c1′*
      have *False*
        by (*fastforce elim*: *isFaultE dest*: *Fault-end simp add*: *final-notin-def*)
      thus *?thesis* **..**
    next
      case *False*
      with *t′-noFault* **have** *t′*: *t′*=*Abrupt t* **by** *simp*
      with *termi-c2′ exec-c1′*
      have *termi-c2′*: $\Gamma\vdash$*c2′*$\downarrow$ *Normal t*
        by *auto*
      with *noFault exec-c1′ t′*
      have $\Gamma\vdash\langle$*c2′*,*Normal t* $\rangle \Rightarrow\notin$*Fault* ‘ *UNIV*
        by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
      from *termi-c2′ c2-c2′ this*
      show $\Gamma\vdash$*c2* $\downarrow$ *Normal t*
        by (*rule Catch.hyps*)
    qed
  qed
}
ultimately show *?case* **using** *c* **by** (*auto intro*: *terminates.intros*)
qed

theorem *terminates-fewer-guards*:
  shows $[\![\Gamma\vdash c'\downarrow s;\ c \subseteq_g c';\ \Gamma\vdash\langle c',s\ \rangle \Rightarrow\notin Fault\ \text{‘}\ UNIV]\!]$

$\Longrightarrow \Gamma\vdash c\downarrow s$
by (*cases s*) (*auto intro*: *terminates-fewer-guards-Normal*)

lemma *terminates-noFault-strip-guards*:
assumes *termi*: $\Gamma\vdash c\downarrow$*Normal s*
shows $[\![\Gamma\vdash\langle c,$*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ $F]\!] \Longrightarrow \Gamma\vdash$*strip-guards F c*$\downarrow$*Normal s*
using *termi*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Basic* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Spec* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Guard s g c f*)
have *s-in-g*: $s \in g$ **by** *fact*
have $\Gamma\vdash c \downarrow$ *Normal s* **by** *fact*
have $\Gamma\vdash\langle$*Guard f g c*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F* **by** *fact*
with *s-in-g* **have** $\Gamma\vdash\langle c,$*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F*
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
with *Guard.hyps* **have** $\Gamma\vdash$*strip-guards F c* $\downarrow$ *Normal s* **by** *simp*
with *s-in-g* **show** *?case*
by (*auto intro*: *terminates.intros*)
next
case *GuardFault* **thus** *?case*
by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *Fault* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Seq c1 s c2*)
have *noFault-Seq*: $\Gamma\vdash\langle$*Seq c1 c2*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F* **by** *fact*
hence *noFault-c1*: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
with *Seq.hyps* **have** $\Gamma\vdash$*strip-guards F c1* $\downarrow$ *Normal s* **by** *simp*
moreover
{
fix s'
assume *exec-strip-guards-c1*: $\Gamma\vdash\langle$*strip-guards F c1*,*Normal s* $\rangle \Rightarrow s'$
have $\Gamma\vdash$*strip-guards F c2* $\downarrow s'$
proof (*cases isFault* s')
case *True*
thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
next
case *False*
with *exec-strip-guards-to-exec* [*OF exec-strip-guards-c1*] *noFault-c1*
have $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this noFault-Seq* **have** $\Gamma\vdash\langle c2,s' \rangle \Rightarrow\notin$*Fault* ‘ *F*

by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *Seq.hyps* **by** *simp*
qed
}
ultimately show *?case*
by (*auto intro*: *terminates.intros*)
next
case *CondTrue* **thus** *?case*
by (*fastforce intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *CondFalse* **thus** *?case*
by (*fastforce intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case (*WhileTrue s b c*)
have *s-in-b*: $s \in b$ **by** *fact*
have *noFault-while*: $\Gamma\vdash\langle$*While b c*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *F* **by** *fact*
with *s-in-b* **have** *noFault-c*: $\Gamma\vdash\langle$*c*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *F*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
with *WhileTrue.hyps* **have** $\Gamma\vdash$*strip-guards F c* $\downarrow$ *Normal s* **by** *simp*
moreover
{
fix s'
assume *exec-strip-guards-c*: $\Gamma\vdash\langle$*strip-guards F c*,*Normal s* $\rangle \Rightarrow s'$
have $\Gamma\vdash$*strip-guards F* (*While b c*) $\downarrow s'$
proof (*cases isFault* s')
case *True*
thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
next
case *False*
with *exec-strip-guards-to-exec* [*OF exec-strip-guards-c*] *noFault-c*
have $\Gamma\vdash\langle$*c*,*Normal s* $\rangle \Rightarrow s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this s-in-b noFault-while* **have** $\Gamma\vdash\langle$*While b c*,s' $\rangle \Rightarrow\notin$*Fault* ' *F*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *WhileTrue.hyps* **by** *simp*
qed
}
ultimately show *?case*
using *WhileTrue.hyps* **by** (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Call* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next

case *Stuck* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case*
by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Abrupt* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Catch c1 s c2*)
have *noFault-Catch*: $\Gamma \vdash \langle Catch\ c1\ c2, Normal\ s\ \rangle \Rightarrow \notin Fault\ `\ F$ **by** *fact*
hence *noFault-c1*: $\Gamma \vdash \langle c1, Normal\ s\ \rangle \Rightarrow \notin Fault\ `\ F$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
with *Catch.hyps* **have** $\Gamma \vdash strip\text{-}guards\ F\ c1 \downarrow Normal\ s$ **by** *simp*
moreover
{
fix s'
assume *exec-strip-guards-c1*: $\Gamma \vdash \langle strip\text{-}guards\ F\ c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
have $\Gamma \vdash strip\text{-}guards\ F\ c2 \downarrow Normal\ s'$
proof −
from *exec-strip-guards-to-exec* [*OF exec-strip-guards-c1*] *noFault-c1*
have $\Gamma \vdash \langle c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this noFault-Catch* **have** $\Gamma \vdash \langle c2, Normal\ s'\ \rangle \Rightarrow \notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *Catch.hyps* **by** *simp*
qed
}
ultimately show *?case*
using *Catch.hyps* **by** (*auto intro*: *terminates.intros*)
qed

3.8 Lemmas about *strip-guards*

lemma *terminates-noFault-strip*:
assumes *termi*: $\Gamma \vdash c \downarrow Normal\ s$
shows $[\![\Gamma \vdash \langle c, Normal\ s\ \rangle \Rightarrow \notin Fault\ `\ F]\!] \Longrightarrow strip\ F\ \Gamma \vdash c \downarrow Normal\ s$
using *termi*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Basic* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Spec* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Guard s g c f*)
have *s-in-g*: $s \in g$ **by** *fact*

```
  have Γ⊢⟨Guard f g c,Normal s ⟩ ⇒∉Fault ‘ F by fact
  with s-in-g have Γ⊢⟨c,Normal s ⟩ ⇒∉Fault ‘ F
    by (fastforce simp add: final-notin-def intro: exec.intros)
  then have strip F Γ⊢c ↓ Normal s by (simp add: Guard.hyps)
  with s-in-g show ?case
    by (auto intro: terminates.intros simp del: strip-simp)
next
  case GuardFault thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case Fault thus ?case by (auto intro: terminates.intros)
next
  case (Seq c1 s c2)
  have noFault-Seq: Γ⊢⟨Seq c1 c2,Normal s ⟩ ⇒∉Fault ‘ F by fact
  hence noFault-c1: Γ⊢⟨c1,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto simp add: final-notin-def intro: exec.intros)
  then have strip F Γ⊢c1 ↓ Normal s by (simp add: Seq.hyps)
  moreover
  {
    fix s'
    assume exec-strip-c1: strip F Γ⊢⟨c1,Normal s ⟩ ⇒ s'
    have strip F Γ⊢c2 ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-to-exec [OF exec-strip-c1] noFault-c1
      have Γ⊢⟨c1,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      moreover
      from this noFault-Seq have Γ⊢⟨c2,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      ultimately show ?thesis
        using Seq.hyps by (simp del: strip-simp)
    qed
  }
  ultimately show ?case
    by (fastforce intro: terminates.intros)
next
  case CondTrue thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case CondFalse thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case (WhileTrue s b c)
  have s-in-b: s ∈ b by fact
  have noFault-while: Γ⊢⟨While b c,Normal s ⟩ ⇒∉Fault ‘ F by fact
```

```
  with s-in-b have noFault-c: Γ⊢⟨c,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto simp add: final-notin-def intro: exec.intros)
  then have strip F Γ⊢c ↓ Normal s by (simp add: WhileTrue.hyps)
  moreover
  {
    fix s'
    assume exec-strip-c: strip F Γ⊢⟨c,Normal s ⟩ ⇒ s'
    have strip F Γ⊢While b c ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-to-exec [OF exec-strip-c] noFault-c
      have Γ⊢⟨c,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      moreover
      from this s-in-b noFault-while have Γ⊢⟨While b c,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      ultimately show ?thesis
        using WhileTrue.hyps by (simp del: strip-simp)
    qed
  }
  ultimately show ?case
    using WhileTrue.hyps by (auto intro: terminates.intros simp del: strip-simp)
next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
next
  case (Call p bdy s)
  have bdy: Γ p = Some bdy by fact
  have Γ⊢⟨Call p,Normal s ⟩ ⇒∉Fault ‘ F by fact
  with bdy have bdy-noFault: Γ⊢⟨bdy,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto intro: exec.intros simp add: final-notin-def)
  then have strip-bdy-noFault: strip F Γ⊢⟨bdy,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto simp add: final-notin-def dest!: exec-strip-to-exec elim!: isFaultE)

  from bdy-noFault have strip F Γ⊢bdy ↓ Normal s by (simp add: Call.hyps)
  from terminates-noFault-strip-guards [OF this strip-bdy-noFault]
  have strip F Γ⊢strip-guards F bdy ↓ Normal s.
  with bdy show ?case
    by (fastforce intro: terminates.Call)
next
  case CallUndefined thus ?case by (auto intro: terminates.intros)
next
  case Stuck thus ?case by (auto intro: terminates.intros)
next
  case DynCom thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
```

case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Abrupt* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Catch c1 s c2*)
have *noFault-Catch*: $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
hence *noFault-c1*: $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
then have *strip F* $\Gamma\vdash c1 \downarrow Normal\ s$ **by** (*simp add*: *Catch.hyps*)
moreover
{
fix s'
assume *exec-strip-c1*: *strip F* $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
have *strip F* $\Gamma\vdash c2 \downarrow Normal\ s'$
proof −
from *exec-strip-to-exec* [*OF exec-strip-c1*] *noFault-c1*
have $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this noFault-Catch* **have** $\Gamma\vdash\langle c2,Normal\ s'\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *Catch.hyps* **by** (*simp del*: *strip-simp*)
qed
}
ultimately show *?case*
using *Catch.hyps* **by** (*auto intro*: *terminates.intros simp del*: *strip-simp*)
qed

3.9 Miscellaneous

lemma *terminates-while-lemma*:
assumes *termi*: $\Gamma\vdash w\downarrow fk$
shows $\bigwedge k\ b\ c.\ [\![fk = Normal\ (f\ k);\ w{=}While\ b\ c;$
$\forall i.\ \Gamma\vdash\langle c,Normal\ (f\ i)\ \rangle \Rightarrow Normal\ (f\ (Suc\ i))]\!]$
$\Longrightarrow \exists i.\ f\ i \notin b$
using *termi*
proof (*induct*)
case *WhileTrue* **thus** *?case* **by** *blast*
next
case *WhileFalse* **thus** *?case* **by** *blast*
qed *simp-all*

lemma *terminates-while*:
$[\![\Gamma\vdash(While\ b\ c)\downarrow Normal\ (f\ k);$
$\forall i.\ \Gamma\vdash\langle c,Normal\ (f\ i)\ \rangle \Rightarrow Normal\ (f\ (Suc\ i))]\!]$
$\Longrightarrow \exists i.\ f\ i \notin b$
by (*blast intro*: *terminates-while-lemma*)

```
lemma wf-terminates-while:
 wf {(t,s). Γ⊢(While b c)↓Normal s ∧ s∈b ∧
            Γ⊢⟨c,Normal s ⟩ ⇒ Normal t}
apply(subst wf-iff-no-infinite-down-chain)
apply(rule notI)
apply clarsimp
apply(insert terminates-while)
apply blast
done

lemma terminates-restrict-to-terminates:
  assumes terminates-res: Γ|M⊢ c ↓ s
  assumes not-Stuck: Γ|M⊢⟨c,s ⟩ ⇒∉{Stuck}
  shows Γ⊢ c ↓ s
using terminates-res not-Stuck
proof (induct)
  case Skip show ?case by (rule terminates.Skip)
next
  case Basic show ?case by (rule terminates.Basic)
next
  case Spec show ?case by (rule terminates.Spec)
next
  case Guard thus ?case
    by (auto intro: terminates.Guard dest: notStuck-GuardD)
next
  case GuardFault thus ?case by (auto intro: terminates.GuardFault)
next
  case Fault show ?case by (rule terminates.Fault)
next
  case (Seq c1 s c2)
  have not-Stuck: Γ|M⊢⟨Seq c1 c2,Normal s ⟩ ⇒∉{Stuck} by fact
  hence c1-notStuck: Γ|M⊢⟨c1,Normal s ⟩ ⇒∉{Stuck}
    by (rule notStuck-SeqD1)
  show Γ⊢Seq c1 c2 ↓ Normal s
  proof (rule terminates.Seq,safe)
    from c1-notStuck
    show Γ⊢c1 ↓ Normal s
      by (rule Seq.hyps)
  next
    fix s'
    assume exec: Γ⊢⟨c1,Normal s ⟩ ⇒ s'
    show Γ⊢c2 ↓ s'
    proof −
      from exec-to-exec-restrict [OF exec] obtain t' where
        exec-res: Γ|M⊢⟨c1,Normal s ⟩ ⇒ t' and
        t'-notStuck: t' ≠ Stuck ⟶ t' = s'
        by blast
      show ?thesis
      proof (cases t'=Stuck)
```

```
        case True
        with c1-notStuck exec-res have False
          by (auto simp add: final-notin-def)
        thus ?thesis ..
      next
        case False
        with t'-notStuck have t': t'=s' by simp
        with not-Stuck exec-res
        have Γ|M⊢⟨c2,s' ⟩ ⇒∉{Stuck}
          by (auto dest: notStuck-SeqD2)
        with exec-res t' Seq.hyps
        show ?thesis
          by auto
      qed
    qed
  qed
next
  case CondTrue thus ?case
    by (auto intro: terminates.CondTrue dest: notStuck-CondTrueD)
next
  case CondFalse thus ?case
    by (auto intro: terminates.CondFalse dest: notStuck-CondFalseD)
next
  case (WhileTrue s b c)
  have s: s ∈ b by fact
  have not-Stuck: Γ|M⊢⟨While b c,Normal s ⟩ ⇒∉{Stuck} by fact
  with WhileTrue have c-notStuck: Γ|M⊢⟨c,Normal s ⟩ ⇒∉{Stuck}
    by (iprover intro: notStuck-WhileTrueD1)
  show ?case
  proof (rule terminates.WhileTrue [OF s],safe)
    from c-notStuck
    show Γ⊢c ↓ Normal s
      by (rule WhileTrue.hyps)
  next
    fix s'
    assume exec: Γ⊢⟨c,Normal s ⟩ ⇒ s'
    show Γ⊢While b c ↓ s'
    proof −
      from exec-to-exec-restrict [OF exec] obtain t' where
        exec-res: Γ|M⊢⟨c,Normal s ⟩ ⇒ t' and
        t'-notStuck: t' ≠ Stuck ⟶ t' = s'
        by blast
      show ?thesis
      proof (cases t'=Stuck)
        case True
        with c-notStuck exec-res have False
          by (auto simp add: final-notin-def)
        thus ?thesis ..
      next
```

```
      case False
      with t'-notStuck have t': t'=s' by simp
      with not-Stuck exec-res s
      have Γ|_M⊢⟨While b c,s' ⟩ ⇒∉{Stuck}
        by (auto dest: notStuck-WhileTrueD2)
      with exec-res t' WhileTrue.hyps
      show ?thesis
        by auto
    qed
  qed
 qed
next
 case WhileFalse then show ?case by (iprover intro: terminates.WhileFalse)
next
 case Call thus ?case
   by (auto intro: terminates.Call dest: notStuck-CallD restrict-SomeD)
next
 case CallUndefined
 thus ?case
   by (auto dest: notStuck-CallDefinedD)
next
 case Stuck show ?case by (rule terminates.Stuck)
next
 case DynCom
 thus ?case
   by (auto intro: terminates.DynCom dest: notStuck-DynComD)
next
 case Throw show ?case by (rule terminates.Throw)
next
 case Abrupt show ?case by (rule terminates.Abrupt)
next
 case (Catch c1 s c2)
 have not-Stuck: Γ|_M⊢⟨Catch c1 c2,Normal s ⟩ ⇒∉{Stuck} by fact
 hence c1-notStuck: Γ|_M⊢⟨c1,Normal s ⟩ ⇒∉{Stuck}
   by (rule notStuck-CatchD1)
 show Γ⊢Catch c1 c2 ↓ Normal s
 proof (rule terminates.Catch,safe)
   from c1-notStuck
   show Γ⊢c1 ↓ Normal s
     by (rule Catch.hyps)
 next
   fix s'
   assume exec: Γ⊢⟨c1,Normal s ⟩ ⇒ Abrupt s'
   show Γ⊢c2 ↓ Normal s'
   proof –
     from exec-to-exec-restrict [OF exec] obtain t' where
       exec-res: Γ|_M⊢⟨c1,Normal s ⟩ ⇒ t' and
       t'-notStuck: t' ≠ Stuck ⟶ t' = Abrupt s'
       by blast
```

show *?thesis*
proof (*cases* t'=*Stuck*)
case *True*
with *c1-notStuck exec-res* **have** *False*
by (*auto simp add*: *final-notin-def*)
thus *?thesis* **..**
next
case *False*
with *t'-notStuck* **have** t': t'=*Abrupt* s' **by** *simp*
with *not-Stuck exec-res*
have $\Gamma|_M\vdash\langle c2, Normal\ s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto dest*: *notStuck-CatchD2*)
with *exec-res* t' *Catch.hyps*
show *?thesis*
by *auto*
qed
qed
qed
qed

end

4 Small-Step Semantics and Infinite Computations

theory *SmallStep* **imports** *Termination*
begin

The redex of a statement is the substatement, which is actually altered by the next step in the small-step semantics.

primrec *redex*:: $('s,'p,'f)com \Rightarrow ('s,'p,'f)com$
where
redex Skip = *Skip* |
redex (*Basic f*) = (*Basic f*) |
redex (*Spec r*) = (*Spec r*) |
redex (*Seq* c_1 c_2) = *redex* c_1 |
redex (*Cond b* c_1 c_2) = (*Cond b* c_1 c_2) |
redex (*While b c*) = (*While b c*) |
redex (*Call p*) = (*Call p*) |
redex (*DynCom d*) = (*DynCom d*) |
redex (*Guard f b c*) = (*Guard f b c*) |
redex (*Throw*) = *Throw* |
redex (*Catch* c_1 c_2) = *redex* c_1

4.1 Small-Step Computation: $\Gamma\vdash(c,\ s) \rightarrow (c',\ s')$

type-synonym $('s,'p,'f)$ *config* = $('s,'p,'f)com \times ('s,'f)$ *xstate*
inductive *step*::[$('s,'p,'f)$ *body*,$('s,'p,'f)$ *config*,$('s,'p,'f)$ *config*] $\Rightarrow$ *bool*
(-$\vdash$ (- $\rightarrow$/ -) [*81*,*81*,*81*] *100*)
for Γ::$('s,'p,'f)$ *body*

where

Basic: $\Gamma\vdash(\mathit{Basic}\ f,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Normal}\ (f\ s))$

$|\ \mathit{Spec}$: $(s,t) \in r \Longrightarrow \Gamma\vdash(\mathit{Spec}\ r,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Normal}\ t)$
$|\ \mathit{SpecStuck}$: $\forall\, t.\ (s,t) \notin r \Longrightarrow \Gamma\vdash(\mathit{Spec}\ r,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Stuck})$

$|\ \mathit{Guard}$: $s{\in}g \Longrightarrow \Gamma\vdash(\mathit{Guard}\ f\ g\ c,\mathit{Normal}\ s) \rightarrow (c,\mathit{Normal}\ s)$

$|\ \mathit{GuardFault}$: $s{\notin}g \Longrightarrow \Gamma\vdash(\mathit{Guard}\ f\ g\ c,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Fault}\ f)$

$|\ \mathit{Seq}$: $\Gamma\vdash(c_1,s) \rightarrow (c_1{}',s')$
$\Longrightarrow$
$\Gamma\vdash(\mathit{Seq}\ c_1\ c_2,s) \rightarrow (\mathit{Seq}\ c_1{}'\ c_2,\ s')$
$|\ \mathit{SeqSkip}$: $\Gamma\vdash(\mathit{Seq}\ \mathit{Skip}\ c_2,s) \rightarrow (c_2,\ s)$
$|\ \mathit{SeqThrow}$: $\Gamma\vdash(\mathit{Seq}\ \mathit{Throw}\ c_2,\mathit{Normal}\ s) \rightarrow (\mathit{Throw},\ \mathit{Normal}\ s)$

$|\ \mathit{CondTrue}$: $s{\in}b \Longrightarrow \Gamma\vdash(\mathit{Cond}\ b\ c_1\ c_2,\mathit{Normal}\ s) \rightarrow (c_1,\mathit{Normal}\ s)$
$|\ \mathit{CondFalse}$: $s{\notin}b \Longrightarrow \Gamma\vdash(\mathit{Cond}\ b\ c_1\ c_2,\mathit{Normal}\ s) \rightarrow (c_2,\mathit{Normal}\ s)$

$|\ \mathit{WhileTrue}$: $[\![s{\in}b]\!]$
$\Longrightarrow$
$\Gamma\vdash(\mathit{While}\ b\ c,\mathit{Normal}\ s) \rightarrow (\mathit{Seq}\ c\ (\mathit{While}\ b\ c),\mathit{Normal}\ s)$

$|\ \mathit{WhileFalse}$: $[\![s{\notin}b]\!]$
$\Longrightarrow$
$\Gamma\vdash(\mathit{While}\ b\ c,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Normal}\ s)$

$|\ \mathit{Call}$: $\Gamma\ p{=}\mathit{Some}\ \mathit{bdy} \Longrightarrow$
$\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ s) \rightarrow (\mathit{bdy},\mathit{Normal}\ s)$

$|\ \mathit{CallUndefined}$: $\Gamma\ p{=}\mathit{None} \Longrightarrow$
$\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Stuck})$

$|\ \mathit{DynCom}$: $\Gamma\vdash(\mathit{DynCom}\ c,\mathit{Normal}\ s) \rightarrow (c\ s,\mathit{Normal}\ s)$

$|\ \mathit{Catch}$: $[\![\Gamma\vdash(c_1,s) \rightarrow (c_1{}',s')]\!]$
$\Longrightarrow$
$\Gamma\vdash(\mathit{Catch}\ c_1\ c_2,s) \rightarrow (\mathit{Catch}\ c_1{}'\ c_2,s')$

$|\ \mathit{CatchThrow}$: $\Gamma\vdash(\mathit{Catch}\ \mathit{Throw}\ c_2,\mathit{Normal}\ s) \rightarrow (c_2,\mathit{Normal}\ s)$
$|\ \mathit{CatchSkip}$: $\Gamma\vdash(\mathit{Catch}\ \mathit{Skip}\ c_2,s) \rightarrow (\mathit{Skip},s)$

$|\ \mathit{FaultProp}$: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash(c,\mathit{Fault}\ f) \rightarrow (\mathit{Skip},\mathit{Fault}\ f)$
$|\ \mathit{StuckProp}$: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash(c,\mathit{Stuck}) \rightarrow (\mathit{Skip},\mathit{Stuck})$
$|\ \mathit{AbruptProp}$: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash(c,\mathit{Abrupt}\ f) \rightarrow (\mathit{Skip},\mathit{Abrupt}\ f)$

lemmas *step-induct* = *step.induct* [*of* - (*c*,*s*) (*c'*,*s'*), *split-format* (*complete*), *case-names* *Basic Spec SpecStuck Guard GuardFault Seq SeqSkip SeqThrow CondTrue CondFalse* *WhileTrue WhileFalse Call CallUndefined DynCom Catch CatchThrow CatchSkip* *FaultProp StuckProp AbruptProp*, *induct set*]

inductive-cases *step-elim-cases* [*cases set*]:
$\Gamma\vdash(Skip,s) \rightarrow u$
$\Gamma\vdash(Guard\ f\ g\ c,s) \rightarrow u$
$\Gamma\vdash(Basic\ f,s) \rightarrow u$
$\Gamma\vdash(Spec\ r,s) \rightarrow u$
$\Gamma\vdash(Seq\ c1\ c2,s) \rightarrow u$
$\Gamma\vdash(Cond\ b\ c1\ c2,s) \rightarrow u$
$\Gamma\vdash(While\ b\ c,s) \rightarrow u$
$\Gamma\vdash(Call\ p,s) \rightarrow u$
$\Gamma\vdash(DynCom\ c,s) \rightarrow u$
$\Gamma\vdash(Throw,s) \rightarrow u$
$\Gamma\vdash(Catch\ c1\ c2,s) \rightarrow u$

inductive-cases *step-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash(Skip,Normal\ s) \rightarrow u$
$\Gamma\vdash(Guard\ f\ g\ c,Normal\ s) \rightarrow u$
$\Gamma\vdash(Basic\ f,Normal\ s) \rightarrow u$
$\Gamma\vdash(Spec\ r,Normal\ s) \rightarrow u$
$\Gamma\vdash(Seq\ c1\ c2,Normal\ s) \rightarrow u$
$\Gamma\vdash(Cond\ b\ c1\ c2,Normal\ s) \rightarrow u$
$\Gamma\vdash(While\ b\ c,Normal\ s) \rightarrow u$
$\Gamma\vdash(Call\ p,Normal\ s) \rightarrow u$
$\Gamma\vdash(DynCom\ c,Normal\ s) \rightarrow u$
$\Gamma\vdash(Throw,Normal\ s) \rightarrow u$
$\Gamma\vdash(Catch\ c1\ c2,Normal\ s) \rightarrow u$

The final configuration is either of the form (*Skip*,-) for normal termination, or (*Throw*, *Normal s*) in case the program was started in a *Normal* state and terminated abruptly. The *Abrupt* state is not used to model abrupt termination, in contrast to the big-step semantics. Only if the program starts in an *Abrupt* states it ends in the same *Abrupt* state.

definition *final*:: ('s,'p,'f) *config* $\Rightarrow$ *bool* **where**
final cfg = (*fst cfg*=*Skip* $\vee$ (*fst cfg*=*Throw* $\wedge$ ($\exists$ *s*. *snd cfg*=*Normal s*)))

abbreviation
step-rtrancl :: [('s,'p,'f) *body*,('s,'p,'f) *config*,('s,'p,'f) *config*] $\Rightarrow$ *bool*
(-$\vdash$ (- $\rightarrow^*$/ -) [*81*,*81*,*81*] *100*)
where
$\Gamma\vdash cf0 \rightarrow^* cf1 \equiv (CONST\ step\ \Gamma)^{**}\ cf0\ cf1$
abbreviation
step-trancl :: [('s,'p,'f) *body*,('s,'p,'f) *config*,('s,'p,'f) *config*] $\Rightarrow$ *bool*
(-$\vdash$ (- $\rightarrow^+$/ -) [*81*,*81*,*81*] *100*)

where
$\Gamma \vdash cf0 \rightarrow^{+} cf1 \equiv (CONST\ step\ \Gamma)^{++}\ cf0\ cf1$

4.2 Structural Properties of Small Step Computations

lemma *redex-not-Seq*: $redex\ c = Seq\ c1\ c2 \Longrightarrow P$
apply (*induct c*)
apply *auto*
done

lemma *no-step-final*:
assumes *step*: $\Gamma \vdash (c,s) \rightarrow (c',s')$
shows *final* $(c,s) \Longrightarrow P$
using *step*
by *induct* (*auto simp add*: *final-def*)

lemma *no-step-final'*:
assumes *step*: $\Gamma \vdash cfg \rightarrow cfg'$
shows *final cfg* $\Longrightarrow P$
using *step*
by (*cases cfg*, *cases cfg'*) (*auto intro*: *no-step-final*)

lemma *step-Abrupt*:
assumes *step*: $\Gamma \vdash (c,\ s) \rightarrow (c',\ s')$
shows $\bigwedge x.\ s = Abrupt\ x \Longrightarrow s' = Abrupt\ x$
using *step*
by (*induct*) *auto*

lemma *step-Fault*:
assumes *step*: $\Gamma \vdash (c,\ s) \rightarrow (c',\ s')$
shows $\bigwedge f.\ s = Fault\ f \Longrightarrow s' = Fault\ f$
using *step*
by (*induct*) *auto*

lemma *step-Stuck*:
assumes *step*: $\Gamma \vdash (c,\ s) \rightarrow (c',\ s')$
shows $\bigwedge f.\ s = Stuck \Longrightarrow s' = Stuck$
using *step*
by (*induct*) *auto*

lemma *SeqSteps*:
assumes *steps*: $\Gamma \vdash cfg_1 \rightarrow^{*}\ cfg_2$
shows $\bigwedge\ c_1\ s\ c_1'\ s'.\ [\![cfg_1 = (c_1,s); cfg_2 = (c_1',s')]\!]$
$\Longrightarrow \Gamma \vdash (Seq\ c_1\ c_2,s) \rightarrow^{*} (Seq\ c_1'\ c_2,\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
case *Refl*
thus *?case*
by *simp*

next
  case $(Trans\ cfg_1\ cfg'')$
  have *step*: $\Gamma\vdash cfg_1 \rightarrow cfg''$ **by** *fact*
  have *steps*: $\Gamma\vdash cfg'' \rightarrow^* cfg_2$ **by** *fact*
  have cfg_1: $cfg_1 = (c_1,\ s)$ **and** cfg_2: $cfg_2 = (c_1',\ s')$ **by** *fact+*
  obtain $c_1''\ s''$ **where** cfg'': $cfg''=(c_1'',s'')$
    by $(cases\ cfg'')$ *auto*
  from *step* $cfg_1\ cfg''$
  have $\Gamma\vdash (c_1,s) \rightarrow (c_1'',s'')$
    by *simp*
  hence $\Gamma\vdash (Seq\ c_1\ c_2,s) \rightarrow (Seq\ c_1''\ c_2,s'')$
    by $(rule\ step.Seq)$
  also from *Trans.hyps* (3) $[OF\ cfg''\ cfg_2]$
  have $\Gamma\vdash (Seq\ c_1''\ c_2,\ s'') \rightarrow^* (Seq\ c_1'\ c_2,\ s')$ **.**
  finally show *?case* **.**
qed

lemma *CatchSteps*:
  assumes *steps*: $\Gamma\vdash cfg_1\rightarrow^* cfg_2$
  shows $\bigwedge c_1\ s\ c_1'\ s'.\ [\![cfg_1 = (c_1,s);\ cfg_2=(c_1',s')]\!]$
    $\Longrightarrow \Gamma\vdash(Catch\ c_1\ c_2,s) \rightarrow^* (Catch\ c_1'\ c_2,\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
  case *Refl*
  thus *?case*
    by *simp*
next
  case $(Trans\ cfg_1\ cfg'')$
  have *step*: $\Gamma\vdash cfg_1 \rightarrow cfg''$ **by** *fact*
  have *steps*: $\Gamma\vdash cfg'' \rightarrow^* cfg_2$ **by** *fact*
  have cfg_1: $cfg_1 = (c_1,\ s)$ **and** cfg_2: $cfg_2 = (c_1',\ s')$ **by** *fact+*
  obtain $c_1''\ s''$ **where** cfg'': $cfg''=(c_1'',s'')$
    by $(cases\ cfg'')$ *auto*
  from *step* $cfg_1\ cfg''$
  have s: $\Gamma\vdash (c_1,s) \rightarrow (c_1'',s'')$
    by *simp*
  hence $\Gamma\vdash (Catch\ c_1\ c_2,s) \rightarrow (Catch\ c_1''\ c_2,s'')$
    by $(rule\ step.Catch)$
  also from *Trans.hyps* (3) $[OF\ cfg''\ cfg_2]$
  have $\Gamma\vdash (Catch\ c_1''\ c_2,\ s'') \rightarrow^* (Catch\ c_1'\ c_2,\ s')$ **.**
  finally show *?case* **.**
qed

lemma *steps-Fault*: $\Gamma\vdash (c,\ Fault\ f) \rightarrow^* (Skip,\ Fault\ f)$
proof $(induct\ c)$
  case $(Seq\ c_1\ c_2)$
  have *steps-*c_1: $\Gamma\vdash (c_1,\ Fault\ f) \rightarrow^* (Skip,\ Fault\ f)$ **by** *fact*
  have *steps-*c_2: $\Gamma\vdash (c_2,\ Fault\ f) \rightarrow^* (Skip,\ Fault\ f)$ **by** *fact*

from *SeqSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Seq* c_1 c_2, *Fault f*) $\rightarrow^*$ (*Seq Skip* c_2, *Fault f*).
also
have $\Gamma\vdash$ (*Seq Skip* c_2, *Fault f*) $\rightarrow$ (c_2, *Fault f*) **by** (*rule SeqSkip*)
also note *steps-*c_2
finally show *?case* **by** *simp*
next
case (*Catch* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash$ (c_1, *Fault f*) $\rightarrow^*$ (*Skip*, *Fault f*) **by** *fact*
from *CatchSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Catch* c_1 c_2, *Fault f*) $\rightarrow^*$ (*Catch Skip* c_2, *Fault f*).
also
have $\Gamma\vdash$ (*Catch Skip* c_2, *Fault f*) $\rightarrow$ (*Skip*, *Fault f*) **by** (*rule CatchSkip*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *step.intros*)+

lemma *steps-Stuck*: $\Gamma\vdash$ (*c*, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*)
proof (*induct c*)
case (*Seq* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash$ (c_1, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
have *steps-*c_2: $\Gamma\vdash$ (c_2, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
from *SeqSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Seq* c_1 c_2, *Stuck*) $\rightarrow^*$ (*Seq Skip* c_2, *Stuck*).
also
have $\Gamma\vdash$ (*Seq Skip* c_2, *Stuck*) $\rightarrow$ (c_2, *Stuck*) **by** (*rule SeqSkip*)
also note *steps-*c_2
finally show *?case* **by** *simp*
next
case (*Catch* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash$ (c_1, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
from *CatchSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Catch* c_1 c_2, *Stuck*) $\rightarrow^*$ (*Catch Skip* c_2, *Stuck*) .
also
have $\Gamma\vdash$ (*Catch Skip* c_2, *Stuck*) $\rightarrow$ (*Skip*, *Stuck*) **by** (*rule CatchSkip*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *step.intros*)+

lemma *steps-Abrupt*: $\Gamma\vdash$ (*c*, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*)
proof (*induct c*)
case (*Seq* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash$ (c_1, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
have *steps-*c_2: $\Gamma\vdash$ (c_2, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
from *SeqSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Seq* c_1 c_2, *Abrupt s*) $\rightarrow^*$ (*Seq Skip* c_2, *Abrupt s*).
also
have $\Gamma\vdash$ (*Seq Skip* c_2, *Abrupt s*) $\rightarrow$ (c_2, *Abrupt s*) **by** (*rule SeqSkip*)
also note *steps-*c_2
finally show *?case* **by** *simp*
next

case (*Catch* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash$ (c_1, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
from *CatchSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash$ (*Catch* c_1 c_2, *Abrupt s*) $\rightarrow^*$ (*Catch Skip* c_2, *Abrupt s*).
also
have $\Gamma\vdash$ (*Catch Skip* c_2, *Abrupt s*) $\rightarrow$ (*Skip*, *Abrupt s*) **by** (*rule CatchSkip*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *step.intros*)+

lemma *step-Fault-prop*:
assumes *step*: $\Gamma\vdash$ (*c*, *s*) $\rightarrow$ (*c'*, *s'*)
shows $\bigwedge f.$ *s*=*Fault f* $\Longrightarrow$ *s'*=*Fault f*
using *step*
by (*induct*) *auto*

lemma *step-Abrupt-prop*:
assumes *step*: $\Gamma\vdash$ (*c*, *s*) $\rightarrow$ (*c'*, *s'*)
shows $\bigwedge x.$ *s*=*Abrupt x* $\Longrightarrow$ *s'*=*Abrupt x*
using *step*
by (*induct*) *auto*

lemma *step-Stuck-prop*:
assumes *step*: $\Gamma\vdash$ (*c*, *s*) $\rightarrow$ (*c'*, *s'*)
shows *s*=*Stuck* $\Longrightarrow$ *s'*=*Stuck*
using *step*
by (*induct*) *auto*

lemma *steps-Fault-prop*:
assumes *step*: $\Gamma\vdash$ (*c*, *s*) $\rightarrow^*$ (*c'*, *s'*)
shows *s*=*Fault f* $\Longrightarrow$ *s'*=*Fault f*
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *simp*
next
case (*Trans c s c'' s''*)
thus *?case*
by (*auto intro*: *step-Fault-prop*)
qed

lemma *steps-Abrupt-prop*:
assumes *step*: $\Gamma\vdash$ (*c*, *s*) $\rightarrow^*$ (*c'*, *s'*)
shows *s*=*Abrupt t* $\Longrightarrow$ *s'*=*Abrupt t*
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *simp*
next
case (*Trans c s c'' s''*)
thus *?case*
by (*auto intro*: *step-Abrupt-prop*)

qed

lemma *steps-Stuck-prop*:
  assumes *step*: $\Gamma\vdash (c,\ s) \rightarrow^* (c',\ s')$
  shows $s{=}Stuck \Longrightarrow s'{=}Stuck$
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl* **thus** *?case* **by** *simp*
next
  case (*Trans c s c'' s''*)
  thus *?case*
    by (*auto intro*: *step-Stuck-prop*)
qed

4.3 Equivalence between Small-Step and Big-Step Semantics

theorem *exec-impl-steps*:
  assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
  shows $\exists\, c'\ t'.\ \Gamma\vdash(c,s) \rightarrow^* (c',t') \wedge$
            (*case t of*
             Abrupt x $\Rightarrow$ *if* $s{=}t$ *then* $c'{=}Skip \wedge t'{=}t$ *else* $c'{=}Throw \wedge t'{=}Normal$
x
             | - $\Rightarrow$ $c'{=}Skip \wedge t'{=}t$)
using *exec*
proof (*induct*)
  case *Skip* **thus** *?case*
    by *simp*
next
  case *Guard* **thus** *?case* **by** (*blast intro*: *step.Guard rtranclp-trans*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *step.GuardFault rtranclp-trans*)
next
  case *FaultProp* **show** *?case* **by** (*fastforce intro*: *steps-Fault*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *step.Basic rtranclp-trans*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *step.Spec rtranclp-trans*)
next
  case *SpecStuck* **thus** *?case* **by** (*fastforce intro*: *step.SpecStuck rtranclp-trans*)
next
  case (*Seq* c_1 *s* s' c_2 *t*)
  have *exec-*c_1: $\Gamma\vdash \langle c_1, Normal\ s\rangle \Rightarrow s'$ **by** *fact*
  have *exec-*c_2: $\Gamma\vdash \langle c_2, s'\rangle \Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases* $\exists\, x.\ s'{=}Abrupt\ x$)
    case *False*
    from *False Seq.hyps* (*2*)
    have $\Gamma\vdash (c_1,\ Normal\ s) \rightarrow^* (Skip,\ s')$
      by (*cases* s') *auto*

hence *seq-*c_1: $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow^*$ (*Seq Skip* c_2, s')
by (*rule SeqSteps*) *auto*
from *Seq.hyps* (*4*) **obtain** c' t' **where**
steps-c_2: $\Gamma\vdash$ (c_2, s') $\rightarrow^*$ (c', t') **and**
t: (*case* t *of*
Abrupt x $\Rightarrow$ *if* $s' = t$ *then* $c' =$ *Skip* $\wedge$ $t' = t$
else $c' =$ *Throw* $\wedge$ $t' =$ *Normal* x
| - $\Rightarrow$ $c' =$ *Skip* $\wedge$ $t' = t$)
by *auto*
note *seq-*c_1
also have $\Gamma\vdash$ (*Seq Skip* c_2, s') $\rightarrow$ (c_2, s') **by** (*rule step.SeqSkip*)
also note *steps-*c_2
finally have $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow^*$ (c', t').
with t *False* **show** *?thesis*
by (*cases* t) *auto*
next
case *True*
then obtain x **where** s': s'=*Abrupt* x
by *blast*
from s' *Seq.hyps* (*2*)
have $\Gamma\vdash$ (c_1, *Normal* s) $\rightarrow^*$ (*Throw*, *Normal* x)
by *auto*
hence *seq-*c_1: $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow^*$ (*Seq Throw* c_2, *Normal* x)
by (*rule SeqSteps*) *auto*
also have $\Gamma\vdash$ (*Seq Throw* c_2, *Normal* x) $\rightarrow$ (*Throw*, *Normal* x)
by (*rule SeqThrow*)
finally have $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow^*$ (*Throw*, *Normal* x).
moreover
from *exec-*c_2 s' **have** t=*Abrupt* x
by (*auto intro*: *Abrupt-end*)
ultimately show *?thesis*
by *auto*
qed
next
case *CondTrue* **thus** *?case* **by** (*blast intro*: *step.CondTrue rtranclp-trans*)
next
case *CondFalse* **thus** *?case* **by** (*blast intro*: *step.CondFalse rtranclp-trans*)
next
case (*WhileTrue* s b c s' t)
have *exec-c*: $\Gamma\vdash$ $\langle c$,*Normal* $s\rangle$ $\Rightarrow$ s' **by** *fact*
have *exec-w*: $\Gamma\vdash$ $\langle$*While* b c,$s'\rangle$ $\Rightarrow$ t **by** *fact*
have b: $s \in b$ **by** *fact*
hence *step*: $\Gamma\vdash$ (*While* b c,*Normal* s) $\rightarrow$ (*Seq* c (*While* b c),*Normal* s)
by (*rule step.WhileTrue*)
show *?case*
proof (*cases* $\exists x.\ s'$=*Abrupt* x)
case *False*
from *False WhileTrue.hyps* (*3*)
have $\Gamma\vdash$ (c, *Normal* s) $\rightarrow^*$ (*Skip*, s')

by (*cases s'*) *auto*
hence *seq-c*: $\Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\rightarrow^*$ (*Seq Skip* (*While b c*), *s'*)
by (*rule SeqSteps*) *auto*
from *WhileTrue.hyps* (*5*) **obtain** *c' t'* **where**
steps-c$_2$: $\Gamma\vdash$ (*While b c*, *s'*) $\rightarrow^*$ (*c'*, *t'*) **and**
t: (*case t of*
Abrupt x $\Rightarrow$ *if s' = t then c' = Skip* $\wedge$ *t' = t*
else c' = Throw $\wedge$ *t' = Normal x*
| - $\Rightarrow$ *c' = Skip* $\wedge$ *t' = t*)
by *auto*
note *step* **also note** *seq-c*
also have $\Gamma\vdash$ (*Seq Skip* (*While b c*), *s'*) $\rightarrow$ (*While b c*, *s'*)
by (*rule step.SeqSkip*)
also note *steps-c*$_2$
finally have $\Gamma\vdash$ (*While b c*, *Normal s*) $\rightarrow^*$ (*c'*, *t'*)**.**
with *t False* **show** *?thesis*
by (*cases t*) *auto*
next
case *True*
then obtain *x* **where** *s'*: *s'=Abrupt x*
by *blast*
note *step*
also
from *s' WhileTrue.hyps* (*3*)
have $\Gamma\vdash$ (*c*, *Normal s*) $\rightarrow^*$ (*Throw*, *Normal x*)
by *auto*
hence
seq-c: $\Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\rightarrow^*$ (*Seq Throw* (*While b c*), *Normal x*)
by (*rule SeqSteps*) *auto*
also have $\Gamma\vdash$ (*Seq Throw* (*While b c*), *Normal x*) $\rightarrow$ (*Throw*, *Normal x*)
by (*rule SeqThrow*)
finally have $\Gamma\vdash$ (*While b c*, *Normal s*) $\rightarrow^*$ (*Throw*, *Normal x*)**.**
moreover
from *exec-w s'* **have** *t=Abrupt x*
by (*auto intro*: *Abrupt-end*)
ultimately show *?thesis*
by *auto*
qed
next
case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *step.WhileFalse rtrancl-trans*)
next
case *Call* **thus** *?case* **by** (*blast intro*: *step.Call rtranclp-trans*)
next
case *CallUndefined* **thus** *?case* **by** (*fastforce intro*: *step.CallUndefined rtranclp-trans*)
next
case *StuckProp* **thus** *?case* **by** (*fastforce intro*: *steps-Stuck*)
next
case *DynCom* **thus** *?case* **by** (*blast intro*: *step.DynCom rtranclp-trans*)

next
  case *Throw* **thus** *?case* **by** *simp*
next
  case *AbruptProp* **thus** *?case* **by** (*fastforce intro*: *steps-Abrupt*)
next
  case (*CatchMatch* c_1 *s* s' c_2 *t*)
  from *CatchMatch.hyps* (*2*)
  have $\Gamma\vdash$ (c_1, *Normal s*) $\rightarrow^*$ (*Throw*, *Normal* s')
    by *simp*
  hence $\Gamma\vdash$ (*Catch* c_1 c_2, *Normal s*) $\rightarrow^*$ (*Catch Throw* c_2, *Normal* s')
    by (*rule CatchSteps*) *auto*
  also have $\Gamma\vdash$ (*Catch Throw* c_2, *Normal* s') $\rightarrow$ (c_2, *Normal* s')
    by (*rule step.CatchThrow*)
  also
  from *CatchMatch.hyps* (*4*) **obtain** c' t' **where**
      steps-c_2: $\Gamma\vdash$ (c_2, *Normal* s') $\rightarrow^*$ (c', t') **and**
      t: (*case t of*
          Abrupt x $\Rightarrow$ *if Normal* s' = *t then* c' = *Skip* $\wedge$ t' = *t*
                    else c' = *Throw* $\wedge$ t' = *Normal x*
          | - $\Rightarrow$ c' = *Skip* $\wedge$ t' = *t*)
      by *auto*
  note *steps-*c_2
  finally show *?case*
    using *t*
    by (*auto split*: *xstate.splits*)
next
  case (*CatchMiss* c_1 *s t* c_2)
  have *t*: $\neg$ *isAbr t* **by** *fact*
  with *CatchMiss.hyps* (*2*)
  have $\Gamma\vdash$ (c_1, *Normal s*) $\rightarrow^*$ (*Skip*, *t*)
    by (*cases t*) *auto*
  hence $\Gamma\vdash$ (*Catch* c_1 c_2, *Normal s*) $\rightarrow^*$ (*Catch Skip* c_2, *t*)
    by (*rule CatchSteps*) *auto*
  also
  have $\Gamma\vdash$ (*Catch Skip* c_2, *t*) $\rightarrow$ (*Skip*, *t*)
    by (*rule step.CatchSkip*)
  finally show *?case*
    using *t*
    by (*fastforce split*: *xstate.splits*)
qed

corollary *exec-impl-steps-Normal*:
  assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow$ *Normal t*
  shows $\Gamma\vdash(c,s) \rightarrow^*$ (*Skip*, *Normal t*)
using *exec-impl-steps* [*OF exec*]
by *auto*

corollary *exec-impl-steps-Normal-Abrupt*:
  assumes *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow$ *Abrupt t*

shows $\Gamma\vdash(c,Normal\ s) \to^* (Throw,\ Normal\ t)$
using *exec-impl-steps* [*OF exec*]
by *auto*

corollary *exec-impl-steps-Abrupt-Abrupt*:
assumes *exec*: $\Gamma\vdash\langle c,Abrupt\ t\rangle \Rightarrow Abrupt\ t$
shows $\Gamma\vdash(c,Abrupt\ t) \to^* (Skip,\ Abrupt\ t)$
using *exec-impl-steps* [*OF exec*]
by *auto*

corollary *exec-impl-steps-Fault*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f$
shows $\Gamma\vdash(c,s) \to^* (Skip,\ Fault\ f)$
using *exec-impl-steps* [*OF exec*]
by *auto*

corollary *exec-impl-steps-Stuck*:
assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Stuck$
shows $\Gamma\vdash(c,s) \to^* (Skip,\ Stuck)$
using *exec-impl-steps* [*OF exec*]
by *auto*

lemma *step-Abrupt-end*:
assumes *step*: $\Gamma\vdash\ (c_1,\ s) \to (c_1',\ s')$
shows $s'=Abrupt\ x \Longrightarrow s=Abrupt\ x$
using *step*
by *induct auto*

lemma *step-Stuck-end*:
assumes *step*: $\Gamma\vdash\ (c_1,\ s) \to (c_1',\ s')$
shows $s'=Stuck \Longrightarrow$
$s=Stuck\ \vee$
$(\exists\, r\ x.\ redex\ c_1 = Spec\ r \wedge s=Normal\ x \wedge (\forall\, t.\ (x,t)\notin r)) \vee$
$(\exists\, p\ x.\ redex\ c_1=Call\ p \wedge s=Normal\ x \wedge \Gamma\ p = None)$
using *step*
by *induct auto*

lemma *step-Fault-end*:
assumes *step*: $\Gamma\vdash\ (c_1,\ s) \to (c_1',\ s')$
shows $s'=Fault\ f \Longrightarrow$
$s=Fault\ f\ \vee$
$(\exists\, g\ c\ x.\ redex\ c_1 = Guard\ f\ g\ c \wedge s=Normal\ x \wedge x \notin g)$
using *step*
by *induct auto*

lemma *exec-redex-Stuck*:
$\Gamma\vdash\langle redex\ c,s\rangle \Rightarrow Stuck \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow Stuck$
proof (*induct c*)

case *Seq*
thus *?case*
by (*cases s*) (*auto intro*: *exec.intros elim*:*exec-elim-cases*)
next
case *Catch*
thus *?case*
by (*cases s*) (*auto intro*: *exec.intros elim*:*exec-elim-cases*)
qed *simp-all*

lemma *exec-redex-Fault*:
$\Gamma\vdash\langle redex\ c,s\rangle \Rightarrow Fault\ f \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f$
proof (*induct c*)
case *Seq*
thus *?case*
by (*cases s*) (*auto intro*: *exec.intros elim*:*exec-elim-cases*)
next
case *Catch*
thus *?case*
by (*cases s*) (*auto intro*: *exec.intros elim*:*exec-elim-cases*)
qed *simp-all*

lemma *step-extend*:
assumes *step*: $\Gamma\vdash(c,s) \rightarrow (c', s')$
shows $\bigwedge t.\ \Gamma\vdash\langle c',s'\rangle \Rightarrow t \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *step*
proof (*induct*)
case *Basic* **thus** *?case*
by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case *Spec* **thus** *?case*
by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case *SpecStuck* **thus** *?case*
by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case *Guard* **thus** *?case*
by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case *GuardFault* **thus** *?case*
by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case ($Seq\ c_1\ s\ c_1{}'\ s'\ c_2$)
have *step*: $\Gamma\vdash (c_1, s) \rightarrow (c_1{}', s')$ **by** *fact*
have *exec'*: $\Gamma\vdash \langle Seq\ c_1{}'\ c_2,s'\rangle \Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Normal x*)
note *s-Normal* = *this*
show *?thesis*

proof (*cases s'*)
  case (*Normal x'*)
  from *exec'* [*simplified Normal*] **obtain** *s''* **where**
    exec-c₁': $\Gamma\vdash \langle c_1', Normal\ x' \rangle \Rightarrow s''$ **and**
    exec-c₂: $\Gamma\vdash \langle c_2, s'' \rangle \Rightarrow t$
    by *cases*
  from *Seq.hyps* (*2*) *Normal exec-c₁' s-Normal*
  have $\Gamma\vdash \langle c_1, Normal\ x \rangle \Rightarrow s''$
    by *simp*
  from *exec.Seq* [*OF this exec-c₂*] *s-Normal*
  show *?thesis* **by** *simp*
next
  case (*Abrupt x'*)
  with *exec'* **have** *t=Abrupt x'*
    by (*auto intro*:*Abrupt-end*)
  moreover
  from *step Abrupt*
  have *s=Abrupt x'*
    by (*auto intro*: *step-Abrupt-end*)
  ultimately
  show *?thesis*
    by (*auto intro*: *exec.intros*)
next
  case (*Fault f*)
  from *step-Fault-end* [*OF step this*] *s-Normal*
  obtain *g c* **where**
    redex-c₁: $redex\ c_1 = Guard\ f\ g\ c$ **and**
    fail: $x \notin g$
    by *auto*
  hence $\Gamma\vdash \langle redex\ c_1, Normal\ x \rangle \Rightarrow Fault\ f$
    by (*auto intro*: *exec.intros*)
  from *exec-redex-Fault* [*OF this*]
  have $\Gamma\vdash \langle c_1, Normal\ x \rangle \Rightarrow Fault\ f$**.**
  moreover from *Fault exec'* **have** *t=Fault f*
    by (*auto intro*: *Fault-end*)
  ultimately
  show *?thesis*
    using *s-Normal*
    by (*auto intro*: *exec.intros*)
next
  case *Stuck*
  from *step-Stuck-end* [*OF step this*] *s-Normal*
  have $(\exists r.\ redex\ c_1 = Spec\ r \wedge (\forall t.\ (x,\ t) \notin r)) \vee$
       $(\exists p.\ redex\ c_1 = Call\ p \wedge \Gamma\ p = None)$
    by *auto*
  moreover
  {
    fix *r*
    assume $redex\ c_1 = Spec\ r$ **and** $(\forall t.\ (x,\ t) \notin r)$

hence $\Gamma\vdash \langle redex\ c_1,Normal\ x\rangle \Rightarrow Stuck$
  by (*auto intro*: *exec.intros*)
from *exec-redex-Stuck* [*OF this*]
have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow Stuck$.
moreover from *Stuck exec′* **have** *t*=*Stuck*
  by (*auto intro*: *Stuck-end*)
ultimately
have *?thesis*
  using *s-Normal*
  by (*auto intro*: *exec.intros*)
}
moreover
{
fix *p*
assume *redex* c_1 = *Call p* **and** $\Gamma\ p = None$
hence $\Gamma\vdash \langle redex\ c_1,Normal\ x\rangle \Rightarrow Stuck$
  by (*auto intro*: *exec.intros*)
from *exec-redex-Stuck* [*OF this*]
have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow Stuck$.
moreover from *Stuck exec′* **have** *t*=*Stuck*
  by (*auto intro*: *Stuck-end*)
ultimately
have *?thesis*
  using *s-Normal*
  by (*auto intro*: *exec.intros*)
}
ultimately show *?thesis*
  by *auto*
qed
next
case (*Abrupt x*)
from *step-Abrupt* [*OF step this*]
have *s′*=*Abrupt x*.
with *exec′*
have *t*=*Abrupt x*
  by (*auto intro*: *Abrupt-end*)
with *Abrupt*
show *?thesis*
  by (*auto intro*: *exec.intros*)
next
case (*Fault f*)
from *step-Fault* [*OF step this*]
have *s′*=*Fault f*.
with *exec′*
have *t*=*Fault f*
  by (*auto intro*: *Fault-end*)
with *Fault*
show *?thesis*
  by (*auto intro*: *exec.intros*)

```
  next
    case Stuck
    from step-Stuck [OF step this]
    have s'=Stuck.
    with exec'
    have t=Stuck
      by (auto intro: Stuck-end)
    with Stuck
    show ?thesis
      by (auto intro: exec.intros)
  qed
next
  case (SeqSkip c2 s t) thus ?case
    by (cases s) (fastforce intro: exec.intros elim: exec-elim-cases)+
next
  case (SeqThrow c2 s t) thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)+
next
  case CondTrue thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CondFalse thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case WhileTrue thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case WhileFalse thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case Call thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CallUndefined thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case DynCom thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case (Catch c1 s c1' s' c2 t)
  have step: Γ⊢ (c1, s) → (c1', s') by fact
  have exec': Γ⊢ ⟨Catch c1' c2,s'⟩ ⇒ t by fact
  show ?case
  proof (cases s)
    case (Normal x)
    note s-Normal = this
    show ?thesis
    proof (cases s')
      case (Normal x')
```

from *exec'* [*simplified Normal*]
show *?thesis*
proof (*cases*)
  fix s''
  assume *exec-c$_1$'*: $\Gamma\vdash \langle c_1',Normal\ x'\rangle \Rightarrow Abrupt\ s''$
  assume *exec-c$_2$*: $\Gamma\vdash \langle c_2,Normal\ s''\rangle \Rightarrow t$
  from *Catch.hyps* (*2*) *Normal exec-c$_1$' s-Normal*
  have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow Abrupt\ s''$
    by *simp*
  from *exec.CatchMatch* [*OF this exec-c$_2$*] *s-Normal*
  show *?thesis* **by** *simp*
next
  assume *exec-c$_1$'*: $\Gamma\vdash \langle c_1',Normal\ x'\rangle \Rightarrow t$
  assume *t*: $\neg$ *isAbr t*
  from *Catch.hyps* (*2*) *Normal exec-c$_1$' s-Normal*
  have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow t$
    by *simp*
  from *exec.CatchMiss* [*OF this t*] *s-Normal*
  show *?thesis* **by** *simp*
qed
next
  case (*Abrupt x'*)
  with *exec'* **have** $t=Abrupt\ x'$
    by (*auto intro*:*Abrupt-end*)
  moreover
  from *step Abrupt*
  have $s=Abrupt\ x'$
    by (*auto intro*: *step-Abrupt-end*)
  ultimately
  show *?thesis*
    by (*auto intro*: *exec.intros*)
next
  case (*Fault f*)
  from *step-Fault-end* [*OF step this*] *s-Normal*
  obtain *g c* **where**
    redex-c$_1$: *redex* c_1 = *Guard f g c* **and**
    fail: $x \notin g$
    by *auto*
  hence $\Gamma\vdash \langle redex\ c_1,Normal\ x\rangle \Rightarrow Fault\ f$
    by (*auto intro*: *exec.intros*)
  from *exec-redex-Fault* [*OF this*]
  have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow Fault\ f$.
  moreover from *Fault exec'* **have** $t=Fault\ f$
    by (*auto intro*: *Fault-end*)
  ultimately
  show *?thesis*
    using *s-Normal*
    by (*auto intro*: *exec.intros*)
next

    case *Stuck*
    from *step-Stuck-end* [*OF step this*] *s-Normal*
    have $(\exists r.\ redex\ c_1 = Spec\ r \wedge (\forall t.\ (x,\ t) \notin r)) \vee$
          $(\exists p.\ redex\ c_1 = Call\ p \wedge \Gamma\ p = None)$
      by *auto*
    moreover
    {
      fix r
      assume $redex\ c_1 = Spec\ r$ **and** $(\forall t.\ (x,\ t) \notin r)$
      hence $\Gamma\vdash \langle redex\ c_1, Normal\ x\rangle \Rightarrow Stuck$
        by (*auto intro*: *exec.intros*)
      from *exec-redex-Stuck* [*OF this*]
      have $\Gamma\vdash \langle c_1, Normal\ x\rangle \Rightarrow Stuck$**.**
      moreover from *Stuck exec'* **have** *t=Stuck*
        by (*auto intro*: *Stuck-end*)
      ultimately
      have *?thesis*
        using *s-Normal*
        by (*auto intro*: *exec.intros*)
    }
    moreover
    {
      fix p
      assume $redex\ c_1 = Call\ p$ **and** $\Gamma\ p = None$
      hence $\Gamma\vdash \langle redex\ c_1, Normal\ x\rangle \Rightarrow Stuck$
        by (*auto intro*: *exec.intros*)
      from *exec-redex-Stuck* [*OF this*]
      have $\Gamma\vdash \langle c_1, Normal\ x\rangle \Rightarrow Stuck$**.**
      moreover from *Stuck exec'* **have** *t=Stuck*
        by (*auto intro*: *Stuck-end*)
      ultimately
      have *?thesis*
        using *s-Normal*
        by (*auto intro*: *exec.intros*)
    }
    ultimately show *?thesis*
      by *auto*
  qed
next
  case (*Abrupt x*)
  from *step-Abrupt* [*OF step this*]
  have *s'=Abrupt x***.**
  with *exec'*
  have *t=Abrupt x*
    by (*auto intro*: *Abrupt-end*)
  with *Abrupt*
  show *?thesis*
    by (*auto intro*: *exec.intros*)
next

case (*Fault f*)
from *step-Fault* [*OF step this*]
have s'=*Fault f*.
with *exec'*
have *t*=*Fault f*
  by (*auto intro*: *Fault-end*)
with *Fault*
show *?thesis*
  by (*auto intro*: *exec.intros*)
next
case *Stuck*
from *step-Stuck* [*OF step this*]
have s'=*Stuck*.
with *exec'*
have *t*=*Stuck*
  by (*auto intro*: *Stuck-end*)
with *Stuck*
show *?thesis*
  by (*auto intro*: *exec.intros*)
qed
next
case *CatchThrow* **thus** *?case*
  by (*fastforce intro*: *exec.intros elim*: *exec-Normal-elim-cases*)
next
case *CatchSkip* **thus** *?case*
  by (*fastforce intro*: *exec.intros elim*: *exec-elim-cases*)
next
case *FaultProp* **thus** *?case*
  by (*fastforce intro*: *exec.intros elim*: *exec-elim-cases*)
next
case *StuckProp* **thus** *?case*
  by (*fastforce intro*: *exec.intros elim*: *exec-elim-cases*)
next
case *AbruptProp* **thus** *?case*
  by (*fastforce intro*: *exec.intros elim*: *exec-elim-cases*)
qed

theorem *steps-Skip-impl-exec*:
assumes *steps*: $\Gamma\vdash(c,s) \rightarrow^* (Skip,t)$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case*
  by (*cases t*) (*auto intro*: *exec.intros*)
next
case (*Trans c s c' s'*)
have $\Gamma\vdash (c, s) \rightarrow (c', s')$ **and** $\Gamma\vdash \langle c',s'\rangle \Rightarrow t$ **by** *fact+*
thus *?case*
  by (*rule step-extend*)

qed

theorem *steps-Throw-impl-exec*:
  assumes *steps*: $\Gamma\vdash(c,s) \rightarrow^* (Throw,Normal\ t)$
  shows $\Gamma\vdash\langle c,s\rangle \Rightarrow Abrupt\ t$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl* **thus** *?case*
    by (*auto intro*: *exec.intros*)
next
  case (*Trans c s c′ s′*)
  have $\Gamma\vdash (c,\ s) \rightarrow (c',\ s')$ **and** $\Gamma\vdash \langle c',s'\rangle \Rightarrow Abrupt\ t$ **by** *fact+*
  thus *?case*
    by (*rule step-extend*)
qed

4.4 Infinite Computations: $\Gamma\vdash(c,\ s) \rightarrow \ldots(\infty)$

definition *inf*:: $('s,'p,'f)$ *body* $\Rightarrow$ $('s,'p,'f)$ *config* $\Rightarrow$ *bool*
 (-⊢ - → . . .′(∞′) [*60*,*80*] *100*) **where**
$\Gamma\vdash\ cfg \rightarrow \ldots(\infty) \equiv (\exists f.\ f\ (0::nat) = cfg \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (i+1)))$

lemma *not-infI*: $[\![\bigwedge f.\ [\![f\ 0 = cfg;\ \bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)]\!] \Longrightarrow False]\!]$
    $\Longrightarrow \neg\Gamma\vdash\ cfg \rightarrow \ldots(\infty)$
  by (*auto simp add*: *inf-def*)

4.5 Equivalence between Termination and the Absence of Infinite Computations

lemma *step-preserves-termination*:
  assumes *step*: $\Gamma\vdash(c,s) \rightarrow (c',s')$
  shows $\Gamma\vdash c{\downarrow}s \Longrightarrow \Gamma\vdash c'{\downarrow}s'$
using *step*
proof (*induct*)
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *SpecStuck* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Guard* **thus** *?case*
    by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Seq* c_1 *s* $c_1{}'$ *s′* c_2) **thus** *?case*
    apply (*cases s*)
    apply   (*cases s′*)
    apply   (*fastforce intro*: *terminates.intros step-extend*

elim: *terminates-Normal-elim-cases*)
apply (*fastforce intro*: *terminates.intros dest*: *step-Abrupt-prop step-Fault-prop step-Stuck-prop*)+
done
next
case (*SeqSkip* c_2 *s*)
thus *?case*
apply (*cases s*)
apply (*fastforce intro*: *terminates.intros exec.intros elim*: *terminates-Normal-elim-cases*)+
done
next
case (*SeqThrow* c_2 *s*)
thus *?case*
by (*fastforce intro*: *terminates.intros exec.intros elim*: *terminates-Normal-elim-cases*)
next
case *CondTrue*
thus *?case*
by (*fastforce intro*: *terminates.intros exec.intros elim*: *terminates-Normal-elim-cases*)
next
case *CondFalse*
thus *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *WhileTrue*
thus *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *WhileFalse*
thus *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *Call*
thus *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *CallUndefined*
thus *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *DynCom*
thus *?case*

```
    by (fastforce intro: terminates.intros
          elim: terminates-Normal-elim-cases )
next
  case (Catch c1 s c1' s' c2) thus ?case
    apply (cases s)
    apply     (cases s')
    apply        (fastforce intro: terminates.intros step-extend
                  elim: terminates-Normal-elim-cases)
    apply (fastforce intro: terminates.intros dest: step-Abrupt-prop
      step-Fault-prop step-Stuck-prop)+
    done
next
  case CatchThrow
  thus ?case
   by (fastforce intro: terminates.intros exec.intros
          elim: terminates-Normal-elim-cases )
next
  case (CatchSkip c2 s)
  thus ?case
    by (cases s) (fastforce intro: terminates.intros)+
next
  case FaultProp thus ?case by (fastforce intro: terminates.intros)
next
  case StuckProp thus ?case by (fastforce intro: terminates.intros)
next
  case AbruptProp thus ?case by (fastforce intro: terminates.intros)
qed

lemma steps-preserves-termination:
  assumes steps: Γ⊢(c,s) →* (c',s')
  shows Γ⊢c↓s ⟹ Γ⊢c'↓s'
using steps
proof (induct rule: rtranclp-induct2 [consumes 1, case-names Refl Trans])
  case Refl thus ?case  .
next
  case Trans
  thus ?case
    by (blast dest: step-preserves-termination)
qed

ML ⟪
  ML-Thms.bind-thm (tranclp-induct2, Split-Rule.split-rule @{context}
    (Rule-Insts.read-instantiate @{context}
      [(((a, 0), Position.none), (aa,ab)), (((b, 0), Position.none), (ba,bb))] []
      @{thm tranclp-induct}));
⟫

lemma steps-preserves-termination':
  assumes steps: Γ⊢(c,s) →+ (c',s')
```

shows $\Gamma\vdash c\downarrow s \Longrightarrow \Gamma\vdash c'\downarrow s'$
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*, *case-names Step Trans*])
case *Step* **thus** *?case* **by** (*blast intro*: *step-preserves-termination*)
next
case *Trans*
thus *?case*
by (*blast dest*: *step-preserves-termination*)
qed

definition *head-com*:: $('s,'p,'f)$ *com* $\Rightarrow$ $('s,'p,'f)$ *com*
where
head-com c =
(*case c of*
Seq c_1 c_2 $\Rightarrow$ c_1
| *Catch* c_1 c_2 $\Rightarrow$ c_1
| - $\Rightarrow$ *c*)

definition *head*:: $('s,'p,'f)$ *config* $\Rightarrow$ $('s,'p,'f)$ *config*
where *head cfg* = (*head-com* (*fst cfg*), *snd cfg*)

lemma *le-Suc-cases*: $[\![\bigwedge i.\ [\![i < k]\!] \Longrightarrow P\ i;\ P\ k]\!] \Longrightarrow \forall i<(Suc\ k).\ P\ i$
apply *clarify*
apply (*case-tac i=k*)
apply *auto*
done

lemma *redex-Seq-False*: $\bigwedge c'\ c''.\ (redex\ c = Seq\ c''\ c') = False$
by (*induct c*) *auto*

lemma *redex-Catch-False*: $\bigwedge c'\ c''.\ (redex\ c = Catch\ c''\ c') = False$
by (*induct c*) *auto*

lemma *infinite-computation-extract-head-Seq*:
assumes *inf-comp*: $\forall i::nat.\ \Gamma\vdash f\ i \rightarrow f\ (i+1)$
assumes *f-0*: $f\ 0 = (Seq\ c_1\ c_2,s)$
assumes *not-fin*: $\forall i<k.\ \neg\ final\ (head\ (f\ i))$
shows $\forall i<k.\ (\exists c'\ s'.\ f\ (i + 1) = (Seq\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash head\ (f\ i) \rightarrow head\ (f\ (i+1))$
(**is** $\forall i<k.\ ?P\ i$)
using *not-fin*
proof (*induct k*)
case *0*
show *?case* **by** *simp*
next

case (*Suc k*)
have *not-fin-Suc*:
$\forall\, i{<}Suc\ k.\ \neg\ final\ (head\ (f\ i))$ **by** *fact*
from *this*[*rule-format*] **have** *not-fin-k*:
$\forall\, i{<}k.\ \neg\ final\ (head\ (f\ i))$
apply *clarify*
apply (*subgoal-tac* $i < Suc\ k$)
apply *blast*
apply *simp*
done

from *Suc.hyps* [*OF this*]
have *hyp*: $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (Seq\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash\ head\ (f\ i) \rightarrow head\ (f\ (i + 1))$.
show *?case*
proof (*rule le-Suc-cases*)
fix *i*
assume $i < k$
then show *?P i*
by (*rule hyp* [*rule-format*])
next
show *?P k*
proof −
from *hyp* [*rule-format*, *of* $k - 1$] *f-0*
obtain $c'\ fs'\ L'\ s'$ **where** *f-k*: $f\ k = (Seq\ c'\ c_2,\ s')$
by (*cases k*) *auto*
from *inf-comp* [*rule-format*, *of k*] *f-k*
have $\Gamma\vdash(Seq\ c'\ c_2,\ s') \rightarrow f\ (k + 1)$
by *simp*
moreover
from *not-fin-Suc* [*rule-format*, *of k*] *f-k*
have $\neg\ final\ (c',s')$
by (*simp add*: *final-def head-def head-com-def*)
ultimately
obtain $c''\ s''$ **where**
$\Gamma\vdash(c',\ s') \rightarrow (c'',\ s'')$ **and**
$f\ (k + 1) = (Seq\ c''\ c_2,\ s'')$
by *cases* (*auto simp add*: *redex-Seq-False final-def*)
with *f-k*
show *?thesis*
by (*simp add*: *head-def head-com-def*)
qed
qed
qed

lemma *infinite-computation-extract-head-Catch*:
assumes *inf-comp*: $\forall\, i{::}nat.\ \Gamma\vdash f\ i \rightarrow f\ (i{+}1)$
assumes *f-0*: $f\ 0 = (Catch\ c_1\ c_2,s)$
assumes *not-fin*: $\forall\, i{<}k.\ \neg\ final\ (head\ (f\ i))$

shows $\forall\, i<k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (Catch\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash head\ (f\ i) \rightarrow head\ (f\ (i+1))$
(**is** $\forall\, i<k.\ ?P\ i$)
using *not-fin*
proof (*induct k*)
case *0*
show *?case* **by** *simp*
next
case (*Suc k*)
have *not-fin-Suc*:
$\forall\, i<Suc\ k.\ \neg\ final\ (head\ (f\ i))$ **by** *fact*
from *this*[*rule-format*] **have** *not-fin-k*:
$\forall\, i<k.\ \neg\ final\ (head\ (f\ i))$
apply *clarify*
apply (*subgoal-tac* $i < Suc\ k$)
apply *blast*
apply *simp*
done

from *Suc.hyps* [*OF this*]
have *hyp*: $\forall\, i<k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (Catch\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash\ head\ (f\ i) \rightarrow head\ (f\ (i + 1))$.
show *?case*
proof (*rule le-Suc-cases*)
fix *i*
assume $i < k$
then show *?P i*
by (*rule hyp* [*rule-format*])
next
show *?P k*
proof −
from *hyp* [*rule-format*, *of* $k - 1$] *f-0*
obtain $c'\ fs'\ L'\ s'$ **where** *f-k*: $f\ k = (Catch\ c'\ c_2,\ s')$
by (*cases k*) *auto*
from *inf-comp* [*rule-format*, *of k*] *f-k*
have $\Gamma\vdash(Catch\ c'\ c_2,\ s') \rightarrow f\ (k + 1)$
by *simp*
moreover
from *not-fin-Suc* [*rule-format*, *of k*] *f-k*
have $\neg\ final\ (c',s')$
by (*simp add*: *final-def head-def head-com-def*)
ultimately
obtain $c''\ s''$ **where**
$\Gamma\vdash(c',\ s') \rightarrow (c'',\ s'')$ **and**
$f\ (k + 1) = (Catch\ c''\ c_2,\ s'')$
by *cases* (*auto simp add*: *redex-Catch-False final-def*)+
with *f-k*
show *?thesis*
by (*simp add*: *head-def head-com-def*)

    qed
  qed
qed

lemma *no-inf-Throw*: $\neg\ \Gamma\vdash(\mathit{Throw},s) \rightarrow \ldots(\infty)$
proof
  assume $\Gamma\vdash\ (\mathit{Throw},\ s) \rightarrow \ldots(\infty)$
  then obtain f **where**
    step [*rule-format*]: $\forall\, i{::}\mathit{nat}.\ \Gamma\vdash f\ i \rightarrow f\ (i{+}1)$ **and**
    f-0: $f\ 0 = (\mathit{Throw},\ s)$
    by (*auto simp add*: *inf-def*)
  from *step* [*of 0*, *simplified f-0*] *step* [*of 1*]
  show *False*
    by *cases* (*auto elim*: *step-elim-cases*)
qed

lemma *split-inf-Seq*:
  assumes *inf-comp*: $\Gamma\vdash(\mathit{Seq}\ c_1\ c_2,s) \rightarrow \ldots(\infty)$
  shows $\Gamma\vdash(c_1,s) \rightarrow \ldots(\infty) \lor$
        $(\exists\, s'.\ \Gamma\vdash(c_1,s) \rightarrow^*\ (\mathit{Skip},s') \land \Gamma\vdash(c_2,s') \rightarrow \ldots(\infty))$
proof −
  from *inf-comp* **obtain** f **where**
    step: $\forall\, i{::}\mathit{nat}.\ \Gamma\vdash f\ i \rightarrow f\ (i{+}1)$ **and**
    f-0: $f\ 0 = (\mathit{Seq}\ c_1\ c_2,\ s)$
    by (*auto simp add*: *inf-def*)
  from *f-0* **have** *head-f-0*: $\mathit{head}\ (f\ 0) = (c_1,s)$
    by (*simp add*: *head-def head-com-def*)
  show *?thesis*
  proof (*cases* $\exists\, i.\ \mathit{final}\ (\mathit{head}\ (f\ i))$)
    case *True*
    define k **where** $k = (\mathit{LEAST}\ i.\ \mathit{final}\ (\mathit{head}\ (f\ i)))$
    have *less-k*: $\forall\, i{<}k.\ \neg\ \mathit{final}\ (\mathit{head}\ (f\ i))$
      apply (*intro allI impI*)
      apply (*unfold k-def*)
      apply (*drule not-less-Least*)
      apply *auto*
      done
    from *infinite-computation-extract-head-Seq* [*OF step f-0 this*]
    obtain *step-head*: $\forall\, i{<}k.\ \Gamma\vdash\ \mathit{head}\ (f\ i) \rightarrow \mathit{head}\ (f\ (i\ +\ 1))$ **and**
          conf: $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i\ +\ 1) = (\mathit{Seq}\ c'\ c_2,\ s'))$
      by *blast*
    from *True*
    have *final-f-k*: $\mathit{final}\ (\mathit{head}\ (f\ k))$
      apply −
      apply (*erule exE*)
      apply (*drule LeastI*)
      apply (*simp add*: *k-def*)
      done
    moreover

from *f-0 conf* [*rule-format*, *of* $k - 1$]
obtain c' s' **where** *f-k*: $f\ k = (\mathit{Seq}\ c'\ c_2,s')$
  by (*cases k*) *auto*
moreover
from *step-head* **have** *steps-head*: $\Gamma\vdash \mathit{head}\ (f\ 0) \rightarrow^* \mathit{head}\ (f\ k)$
proof (*induct k*)
  case *0* **thus** *?case* **by** *simp*
next
  case (*Suc m*)
  have *step*: $\forall\, i<\mathit{Suc}\ m.\ \Gamma\vdash\ \mathit{head}\ (f\ i) \rightarrow \mathit{head}\ (f\ (i + 1))$ **by** *fact*
  hence $\forall\, i<m.\ \Gamma\vdash\ \mathit{head}\ (f\ i) \rightarrow \mathit{head}\ (f\ (i + 1))$
    by *auto*
  hence $\Gamma\vdash\ \mathit{head}\ (f\ 0) \rightarrow^*\ \mathit{head}\ (f\ m)$
    by (*rule Suc.hyps*)
  also from *step* [*rule-format*, *of m*]
  have $\Gamma\vdash\ \mathit{head}\ (f\ m) \rightarrow \mathit{head}\ (f\ (m + 1))$ **by** *simp*
  finally show *?case* **by** *simp*
qed
{
  assume *f-k*: $f\ k = (\mathit{Seq}\ \mathit{Skip}\ c_2,\ s')$
  with *steps-head*
  have $\Gamma\vdash(c_1,s) \rightarrow^* (\mathit{Skip},s')$
    using *head-f-0*
    by (*simp add*: *head-def head-com-def*)
  moreover
  from *step* [*rule-format*, *of k*] *f-k*
  obtain $\Gamma\vdash(\mathit{Seq}\ \mathit{Skip}\ c_2,s') \rightarrow (c_2,s')$ **and**
    f-Suc-k: $f\ (k + 1) = (c_2,s')$
    by (*fastforce elim*: *step.cases intro*: *step.intros*)
  define *g* **where***g* $i = f\ (i + (k + 1))$ **for** *i*
  from *f-Suc-k*
  have *g-0*: $g\ 0 = (c_2,s')$
    by (*simp add*: *g-def*)
  from *step*
  have $\forall\, i.\ \Gamma\vdash g\ i \rightarrow g\ (i + 1)$
    by (*simp add*: *g-def*)
  with *g-0* **have** $\Gamma\vdash(c_2,s') \rightarrow \ldots(\infty)$
    by (*auto simp add*: *inf-def*)
  ultimately
  have *?thesis*
    by *auto*
}
moreover
{
  fix *x*
  assume s': $s'=\mathit{Normal}\ x$ **and** *f-k*: $f\ k = (\mathit{Seq}\ \mathit{Throw}\ c_2,\ s')$
  from *step* [*rule-format*, *of k*] *f-k* s'
  obtain $\Gamma\vdash(\mathit{Seq}\ \mathit{Throw}\ c_2,s') \rightarrow (\mathit{Throw},s')$ **and**
    f-Suc-k: $f\ (k + 1) = (\mathit{Throw},s')$

```
        by (fastforce elim: step-elim-cases intro: step.intros)
      define g where g i = f (i + (k + 1)) for i
      from f-Suc-k
      have g-0: g 0 = (Throw,s')
        by (simp add: g-def)
      from step
      have ∀ i. Γ⊢g i → g (i + 1)
        by (simp add: g-def)
      with g-0 have Γ⊢(Throw,s') → ...(∞)
        by (auto simp add: inf-def)
      with no-inf-Throw
      have ?thesis
        by auto
    }
    ultimately
    show ?thesis
      by (auto simp add: final-def head-def head-com-def)
  next
    case False
    then have not-fin: ∀ i. ¬ final (head (f i))
      by blast
    have ∀ i. Γ⊢head (f i) → head (f (i + 1))
    proof
      fix k
      from not-fin
      have ∀ i<(Suc k). ¬ final (head (f i))
        by simp

      from infinite-computation-extract-head-Seq [OF step f-0 this ]
      show Γ⊢ head (f k) → head (f (k + 1)) by simp
    qed
    with head-f-0 have Γ⊢(c₁,s) → ...(∞)
      by (auto simp add: inf-def)
    thus ?thesis
      by simp
  qed
qed

lemma split-inf-Catch:
  assumes inf-comp: Γ⊢(Catch c₁ c₂,s) → ...(∞)
  shows Γ⊢(c₁,s) → ...(∞) ∨
        (∃ s'. Γ⊢(c₁,s) →* (Throw,Normal s') ∧ Γ⊢(c₂,Normal s') → ...(∞))
proof −
  from inf-comp obtain f where
    step: ∀ i::nat. Γ⊢f i → f (i+1) and
    f-0: f 0 = (Catch c₁ c₂, s)
    by (auto simp add: inf-def)
  from f-0 have head-f-0: head (f 0) = (c₁,s)
    by (simp add: head-def head-com-def)
```

```
show ?thesis
proof (cases ∃ i. final (head (f i)))
  case True
  define k where k = (LEAST i. final (head (f i)))
  have less-k: ∀ i<k. ¬ final (head (f i))
    apply (intro allI impI)
    apply (unfold k-def)
    apply (drule not-less-Least)
    apply auto
    done
  from infinite-computation-extract-head-Catch [OF step f-0 this]
  obtain step-head: ∀ i<k. Γ⊢ head (f i) → head (f (i + 1)) and
         conf: ∀ i<k. (∃ c' s'. f (i + 1) = (Catch c' c₂, s'))
    by blast
  from True
  have final-f-k: final (head (f k))
    apply −
    apply (erule exE)
    apply (drule LeastI)
    apply (simp add: k-def)
    done
  moreover
  from f-0 conf [rule-format, of k − 1]
  obtain c' s' where f-k: f k = (Catch c' c₂,s')
    by (cases k) auto
  moreover
  from step-head have steps-head: Γ⊢head (f 0) →* head (f k)
  proof (induct k)
    case 0 thus ?case by simp
  next
    case (Suc m)
    have step: ∀ i<Suc m. Γ⊢ head (f i) → head (f (i + 1)) by fact
    hence ∀ i<m. Γ⊢ head (f i) → head (f (i + 1))
      by auto
    hence Γ⊢ head (f 0) →*  head (f m)
      by (rule Suc.hyps)
    also from step [rule-format, of m]
    have Γ⊢ head (f m) → head (f (m + 1)) by simp
    finally show ?case by simp
  qed
  {
    assume f-k: f k = (Catch Skip c₂, s')
    with steps-head
    have Γ⊢(c₁,s) →* (Skip,s')
      using head-f-0
      by (simp add: head-def head-com-def)
    moreover
    from step [rule-format, of k] f-k
    obtain Γ⊢(Catch Skip c₂,s') → (Skip,s') and
```

f-Suc-k: $f\ (k + 1) = (Skip,s')$
by (*fastforce elim*: *step.cases intro*: *step.intros*)
from *step* [*rule-format*, *of* $k+1$, *simplified f-Suc-k*]
have *?thesis*
by (*rule no-step-final'*) (*auto simp add*: *final-def*)
}
moreover
{
fix x
assume s': s'=*Normal* x **and** *f-k*: $f\ k = (Catch\ Throw\ c_2,\ s')$
with *steps-head*
have $\Gamma\vdash(c_1,s) \rightarrow^* (Throw,s')$
using *head-f-0*
by (*simp add*: *head-def head-com-def*)
moreover
from *step* [*rule-format*, *of* k] *f-k* s'
obtain $\Gamma\vdash(Catch\ Throw\ c_2,s') \rightarrow (c_2,s')$ **and**
f-Suc-k: $f\ (k + 1) = (c_2,s')$
by (*fastforce elim*: *step-elim-cases intro*: *step.intros*)
define g **where** $g\ i = f\ (i + (k + 1))$ **for** i
from *f-Suc-k*
have *g-0*: $g\ 0 = (c_2,s')$
by (*simp add*: *g-def*)
from *step*
have $\forall i.\ \Gamma\vdash g\ i \rightarrow g\ (i + 1)$
by (*simp add*: *g-def*)
with *g-0* **have** $\Gamma\vdash(c_2,s') \rightarrow \ldots(\infty)$
by (*auto simp add*: *inf-def*)
ultimately
have *?thesis*
using s'
by *auto*
}
ultimately
show *?thesis*
by (*auto simp add*: *final-def head-def head-com-def*)
next
case *False*
then have *not-fin*: $\forall i.\ \neg$ *final* (*head* ($f\ i$))
by *blast*
have $\forall i.\ \Gamma\vdash head\ (f\ i) \rightarrow head\ (f\ (i + 1))$
proof
fix k
from *not-fin*
have $\forall i<(Suc\ k).\ \neg$ *final* (*head* ($f\ i$))
by *simp*

from *infinite-computation-extract-head-Catch* [*OF step f-0 this*]
show $\Gamma\vdash$ *head* ($f\ k$) $\rightarrow$ *head* ($f\ (k + 1)$) **by** *simp*

qed
with *head-f-0* **have** $\Gamma\vdash(c_1,s) \rightarrow \ldots(\infty)$
by (*auto simp add*: *inf-def*)
thus *?thesis*
by *simp*
qed
qed

lemma *Skip-no-step*: $\Gamma\vdash(\mathit{Skip},s) \rightarrow \mathit{cfg} \Longrightarrow P$
apply (*erule no-step-final′*)
apply (*simp add*: *final-def*)
done

lemma *not-inf-Stuck*: $\neg\ \Gamma\vdash(c,\mathit{Stuck}) \rightarrow \ldots(\infty)$
proof (*induct c*)
case *Skip*
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (\mathit{Suc}\ i)$
assume *f-0*: $f\ 0 = (\mathit{Skip},\ \mathit{Stuck})$
from *f-step* [*of 0*] *f-0*
show *False*
by (*auto elim*: *Skip-no-step*)
qed
next
case (*Basic g*)
thus *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (\mathit{Suc}\ i)$
assume *f-0*: $f\ 0 = (\mathit{Basic}\ g,\ \mathit{Stuck})$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Spec r*)
thus *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (\mathit{Suc}\ i)$
assume *f-0*: $f\ 0 = (\mathit{Spec}\ r,\ \mathit{Stuck})$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Seq* $c_1\ c_2$)

show *?case*
proof
assume $\Gamma\vdash$ (*Seq* c_1 c_2, *Stuck*) $\rightarrow \ldots(\infty)$
from *split-inf-Seq* [*OF this*] *Seq.hyps*
show *False*
by (*auto dest*: *steps-Stuck-prop*)
qed
next
case (*Cond* b c_1 c_2)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Cond\ b\ c_1\ c_2,\ Stuck)$
from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*While* b c)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (While\ b\ c,\ Stuck)$
from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Call* p)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Call\ p,\ Stuck)$
from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*DynCom* d)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (DynCom\ d,\ Stuck)$
from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
show *False*

by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Guard m g c*)
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Guard\ m\ g\ c,\ Stuck)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case *Throw*
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Throw,\ Stuck)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Catch* c_1 c_2)
show *?case*
proof
assume $\Gamma\vdash$ (*Catch* c_1 c_2, *Stuck*) $\rightarrow \ldots(\infty)$
from *split-inf-Catch* [*OF this*] *Catch.hyps*
show *False*
by (*auto dest*: *steps-Stuck-prop*)
qed
qed

lemma *not-inf-Fault*: $\neg\ \Gamma\vdash(c,Fault\ x) \rightarrow \ldots(\infty)$
proof (*induct c*)
case *Skip*
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Skip,\ Fault\ x)$
from *f-step* [*of 0*] *f-0*
show *False*
by (*auto elim*: *Skip-no-step*)
qed
next
case (*Basic g*)

thus *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Basic\ g,\ Fault\ x)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Spec r*)
thus *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Spec\ r,\ Fault\ x)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case ($Seq\ c_1\ c_2$)
show *?case*
proof
assume $\Gamma \vdash\ (Seq\ c_1\ c_2,\ Fault\ x) \rightarrow \ldots(\infty)$
from *split-inf-Seq* [*OF this*] *Seq.hyps*
show *False*
by (*auto dest*: *steps-Fault-prop*)
qed
next
case ($Cond\ b\ c_1\ c_2$)
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Cond\ b\ c_1\ c_2,\ Fault\ x)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*While b c*)
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (While\ b\ c,\ Fault\ x)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*

    by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*Call p*)
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Call\ p,\ Fault\ x)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*DynCom d*)
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (DynCom\ d,\ Fault\ x)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*Guard m g c*)
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Guard\ m\ g\ c,\ Fault\ x)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case *Throw*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Throw,\ Fault\ x)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*Catch* c_1 c_2)
  show *?case*

proof
  assume $\Gamma\vdash$ $(Catch\ c_1\ c_2,\ Fault\ x) \rightarrow \ldots(\infty)$
  from *split-inf-Catch* [*OF this*] *Catch.hyps*
  show *False*
    by (*auto dest*: *steps-Fault-prop*)
qed
qed

lemma *not-inf-Abrupt*: $\neg\ \Gamma\vdash(c, Abrupt\ s) \rightarrow \ldots(\infty)$
proof (*induct c*)
  case *Skip*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Skip,\ Abrupt\ s)$
    from *f-step* [*of 0*] *f-0*
    show *False*
      by (*auto elim*: *Skip-no-step*)
  qed
next
  case (*Basic g*)
  thus *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Basic\ g,\ Abrupt\ s)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*Spec r*)
  thus *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = (Spec\ r,\ Abrupt\ s)$
    from *f-step* [*of 0*] *f-0* *f-step* [*of 1*]
    show *False*
      by (*fastforce elim*: *Skip-no-step step-elim-cases*)
  qed
next
  case (*Seq* c_1 c_2)
  show *?case*
  proof
    assume $\Gamma\vdash$ $(Seq\ c_1\ c_2,\ Abrupt\ s) \rightarrow \ldots(\infty)$
    from *split-inf-Seq* [*OF this*] *Seq.hyps*
    show *False*

by (*auto dest*: *steps-Abrupt-prop*)
qed
next
case (*Cond* b c_1 c_2)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Cond\ b\ c_1\ c_2,\ Abrupt\ s)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*While* b c)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (While\ b\ c,\ Abrupt\ s)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Call* p)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Call\ p,\ Abrupt\ s)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*DynCom* d)
show *?case*
proof (*rule not-infI*)
fix f
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (DynCom\ d,\ Abrupt\ s)$
from *f-step* [*of 0*] *f-0 f-step* [*of 1*]
show *False*
by (*fastforce elim*: *Skip-no-step step-elim-cases*)
qed
next
case (*Guard* m g c)
show *?case*

```
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Guard m g c, Abrupt s)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case Throw
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Throw, Abrupt s)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Catch c1 c2)
  show ?case
  proof
    assume Γ⊢ (Catch c1 c2, Abrupt s) → ...(∞)
    from split-inf-Catch [OF this] Catch.hyps
    show False
      by (auto dest: steps-Abrupt-prop)
  qed
qed


theorem terminates-impl-no-infinite-computation:
  assumes termi: Γ⊢c ↓ s
  shows ¬ Γ⊢(c,s) → ...(∞)
using termi
proof (induct)
  case (Skip s) thus ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Skip, Normal s)
    from f-step [of 0] f-0
    show False
      by (auto elim: Skip-no-step)
  qed
next
  case (Basic g s)
  thus ?case
  proof (rule not-infI)
```

```
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Basic g, Normal s)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Spec r s)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Spec r, Normal s)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Guard s g c m)
  have g: s ∈ g by fact
  have hyp: ¬ Γ⊢ (c, Normal s) → . . .(∞) by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Guard m g c, Normal s)
    from f-step [of 0] f-0  g
    have f 1 = (c,Normal s)
      by (fastforce elim: step-elim-cases)
    with f-step
    have Γ⊢ (c, Normal s) → . . .(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp show False ..
  qed
next
  case (GuardFault s g m c)
  have g: s ∉ g by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Guard m g c, Normal s)
    from g f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
```

next
  case (*Fault c m*)
  thus *?case*
    by (*rule not-inf-Fault*)
next
  case (*Seq* c_1 *s* c_2)
  show *?case*
  proof
    assume $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal s*) $\to \ldots(\infty)$
    from *split-inf-Seq* [*OF this*] *Seq.hyps*
    show *False*
      by (*auto intro*: *steps-Skip-impl-exec*)
  qed
next
  case (*CondTrue s b c1 c2*)
  have *b*: $s \in b$ **by** *fact*
  have *hyp-c1*: $\neg$ $\Gamma\vdash$ (*c1*, *Normal s*) $\to \ldots(\infty)$ **by** *fact*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.$ $\Gamma\vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: *f 0* = (*Cond b c1 c2*, *Normal s*)
    from *b f-step* [*of 0*] *f-0*
    have *f 1* = (*c1*,*Normal s*)
      by (*auto elim*: *step-Normal-elim-cases*)
    with *f-step*
    have $\Gamma\vdash$ (*c1*, *Normal s*) $\to \ldots(\infty)$
      apply (*simp add*: *inf-def*)
      apply (*rule-tac x*=$\lambda i.$ *f* (*Suc i*) **in** *exI*)
      by *simp*
    with *hyp-c1* **show** *False* **by** *simp*
  qed
next
  case (*CondFalse s b c2 c1*)
  have *b*: $s \notin b$ **by** *fact*
  have *hyp-c2*: $\neg$ $\Gamma\vdash$ (*c2*, *Normal s*) $\to \ldots(\infty)$ **by** *fact*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.$ $\Gamma\vdash f\ i \to f\ (Suc\ i)$
    assume *f-0*: *f 0* = (*Cond b c1 c2*, *Normal s*)
    from *b f-step* [*of 0*] *f-0*
    have *f 1* = (*c2*,*Normal s*)
      by (*auto elim*: *step-Normal-elim-cases*)
    with *f-step*
    have $\Gamma\vdash$ (*c2*, *Normal s*) $\to \ldots(\infty)$
      apply (*simp add*: *inf-def*)
      apply (*rule-tac x*=$\lambda i.$ *f* (*Suc i*) **in** *exI*)
      by *simp*

```
    with hyp-c2 show False by simp
  qed
next
  case (WhileTrue s b c)
  have b: s ∈ b by fact
  have hyp-c: ¬ Γ⊢ (c, Normal s) → …(∞) by fact
  have hyp-w: ∀ s'. Γ⊢ ⟨c,Normal s⟩ ⇒ s' ⟶
                  Γ⊢ While b c ↓ s' ∧ ¬ Γ⊢ (While b c, s') → …(∞) by fact
  have not-inf-Seq: ¬ Γ⊢ (Seq c (While b c), Normal s) → …(∞)
  proof
    assume Γ⊢ (Seq c (While b c), Normal s) → …(∞)
    from split-inf-Seq [OF this] hyp-c hyp-w show False
      by (auto intro: steps-Skip-impl-exec)
  qed
  show ?case
  proof
    assume Γ⊢ (While b c, Normal s) → …(∞)
    then obtain f where
      f-step: ⋀i. Γ⊢f i → f (Suc i) and
      f-0: f 0 = (While b c, Normal s)
      by (auto simp add: inf-def)
    from f-step [of 0] f-0 b
    have f 1 = (Seq c (While b c),Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have Γ⊢ (Seq c (While b c), Normal s) → …(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with not-inf-Seq show False by simp
  qed
next
  case (WhileFalse s b c)
  have b: s ∉ b by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (While b c, Normal s)
    from b f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Call p bdy s)
  have bdy: Γ p = Some bdy by fact
  have hyp: ¬ Γ⊢ (bdy, Normal s) → …(∞) by fact
  show ?case
  proof (rule not-infI)
```

```
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Call p, Normal s)
    from bdy f-step [of 0] f-0
    have f 1 = (bdy,Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have Γ⊢ (bdy, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp show False by simp
  qed
next
  case (CallUndefined p s)
  have no-bdy: Γ p = None by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Call p, Normal s)
    from no-bdy f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Stuck c)
  show ?case
    by (rule not-inf-Stuck)
next
  case (DynCom c s)
  have hyp: ¬ Γ⊢ (c s, Normal s) → ...(∞) by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (DynCom c, Normal s)
    from f-step [of 0] f-0
    have f (Suc 0) = (c s, Normal s)
      by (auto elim: step-elim-cases)
    with f-step have Γ⊢ (c s, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp
    show False by simp
  qed
next
  case (Throw s) thus ?case
```

proof (*rule not-infI*)
  fix *f*
  assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \to f\ (Suc\ i)$
  assume *f-0*: $f\ 0 = (Throw,\ Normal\ s)$
  from *f-step* [*of 0*] *f-0*
  show *False*
    by (*auto elim*: *step-elim-cases*)
qed
next
case (*Abrupt c*)
show *?case*
  by (*rule not-inf-Abrupt*)
next
case (*Catch* c_1 *s* c_2)
show *?case*
proof
  assume $\Gamma\vdash$ (*Catch* c_1 c_2, *Normal s*) $\to \ldots(\infty)$
  from *split-inf-Catch* [*OF this*] *Catch.hyps*
  show *False*
    by (*auto intro*: *steps-Throw-impl-exec*)
qed
qed

definition
termi-call-steps :: ('s,'p,'f) *body* $\Rightarrow$ (('s $\times$ 'p) $\times$ ('s $\times$ 'p))*set*
where
termi-call-steps Γ =
{((t,q),(s,p)). $\Gamma\vdash$*Call p*$\downarrow$*Normal s* $\wedge$
  ($\exists$ c. $\Gamma\vdash$(*Call p*,*Normal s*) $\to^+$ (c,*Normal t*) $\wedge$ *redex c* = *Call q*)}

primrec *subst-redex*:: ('s,'p,'f)*com* $\Rightarrow$ ('s,'p,'f)*com* $\Rightarrow$ ('s,'p,'f)*com*
where
subst-redex Skip c = *c* |
subst-redex (*Basic f*) *c* = *c* |
subst-redex (*Spec r*) *c* = *c* |
subst-redex (*Seq* c_1 c_2) *c* = *Seq* (*subst-redex* c_1 *c*) c_2 |
subst-redex (*Cond b* c_1 c_2) *c* = *c* |
subst-redex (*While b c'*) *c* = *c* |
subst-redex (*Call p*) *c* = *c* |
subst-redex (*DynCom d*) *c* = *c* |
subst-redex (*Guard f b c'*) *c* = *c* |
subst-redex (*Throw*) *c* = *c* |
subst-redex (*Catch* c_1 c_2) *c* = *Catch* (*subst-redex* c_1 *c*) c_2

lemma *subst-redex-redex*:
  subst-redex c (*redex c*) = *c*
  by (*induct c*) *auto*

lemma *redex-subst-redex*: *redex* (*subst-redex c r*) = *redex r*
  by (*induct c*) *auto*

lemma *step-redex'*:
  shows $\Gamma\vdash$(*redex c*,*s*) $\rightarrow$ (r',s') $\Longrightarrow$ $\Gamma\vdash$(*c*,*s*) $\rightarrow$ (*subst-redex c* r',s')
by (*induct c*) (*auto intro*: *step.Seq step.Catch*)

lemma *step-redex*:
  shows $\Gamma\vdash$(*r*,*s*) $\rightarrow$ (r',s') $\Longrightarrow$ $\Gamma\vdash$(*subst-redex c r*,*s*) $\rightarrow$ (*subst-redex c* r',s')
by (*induct c*) (*auto intro*: *step.Seq step.Catch*)

lemma *steps-redex*:
  assumes *steps*: $\Gamma\vdash$ (*r*, *s*) $\rightarrow^*$ (r', s')
  shows $\bigwedge c$. $\Gamma\vdash$(*subst-redex c r*,*s*) $\rightarrow^*$ (*subst-redex c* r',s')
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl*
  show $\Gamma\vdash$ (*subst-redex c* r', s') $\rightarrow^*$ (*subst-redex c* r', s')
    by *simp*
next
  case (*Trans r s* r'' s'')
  have $\Gamma\vdash$ (*r*, *s*) $\rightarrow$ (r'', s'') **by** *fact*
  from *step-redex* [*OF this*]
  have $\Gamma\vdash$ (*subst-redex c r*, *s*) $\rightarrow$ (*subst-redex c* r'', s'').
  also
  have $\Gamma\vdash$ (*subst-redex c* r'', s'') $\rightarrow^*$ (*subst-redex c* r', s') **by** *fact*
  finally show *?case* **.**
qed

ML ⟪
  ML-Thms.bind-thm (*trancl-induct2*, *Split-Rule.split-rule* @{*context*}
    (*Rule-Insts.read-instantiate* @{*context*}
      [(((*a*, *0*), *Position.none*), (*aa*, *ab*)), (((*b*, *0*), *Position.none*), (*ba*, *bb*))] []
      @{*thm trancl-induct*}));
⟫

lemma *steps-redex'*:
  assumes *steps*: $\Gamma\vdash$ (*r*, *s*) $\rightarrow^+$ (r', s')
  shows $\bigwedge c$. $\Gamma\vdash$(*subst-redex c r*,*s*) $\rightarrow^+$ (*subst-redex c* r',s')
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*,*case-names Step Trans*])
  case (*Step* r' s')
  have $\Gamma\vdash$ (*r*, *s*) $\rightarrow$ (r', s') **by** *fact*
  then have $\Gamma\vdash$ (*subst-redex c r*, *s*) $\rightarrow$ (*subst-redex c* r', s')
    by (*rule step-redex*)
  then show $\Gamma\vdash$ (*subst-redex c r*, *s*) $\rightarrow^+$ (*subst-redex c* r', s')**..**
next

case (*Trans* r' s' r'' s'')
have $\Gamma\vdash$ (*subst-redex* c r, s) $\rightarrow^+$ (*subst-redex* c r', s') **by** *fact*
also
have $\Gamma\vdash$ (r', s') $\rightarrow$ (r'', s'') **by** *fact*
hence $\Gamma\vdash$ (*subst-redex* c r', s') $\rightarrow$ (*subst-redex* c r'', s'')
by (*rule step-redex*)
finally show $\Gamma\vdash$ (*subst-redex* c r, s) $\rightarrow^+$ (*subst-redex* c r'', s'') **.**
qed

primrec *seq*:: ($nat \Rightarrow$ ($'s$,$'p$,$'f$)*com*) $\Rightarrow$ $'p \Rightarrow nat \Rightarrow$ ($'s$,$'p$,$'f$)*com*
where
seq c p 0 = *Call p* |
seq c p (*Suc i*) = *subst-redex* (*seq c p i*) (*c i*)

lemma *renumber'*:
assumes *f*: $\forall i.$ (a,f i) $\in r^*$ $\wedge$ (f i,f(*Suc* i)) $\in r$
assumes *a-b*: (a,b) $\in r^*$
shows $b = f\ 0 \Longrightarrow (\exists f.\ f\ 0 = a \wedge (\forall i.\ (f\ i,\ f(Suc\ i)) \in r))$
using *a-b*
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
assume $b = f\ 0$
with *f* **show** $\exists f.\ f\ 0 = b \wedge (\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r)$
by *blast*
next
fix *a z*
assume *a-z*: (a, z) $\in r$ **and** (z, b) $\in r^*$
assume $b = f\ 0 \Longrightarrow \exists f.\ f\ 0 = z \wedge (\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r)$
$b = f\ 0$
then obtain *f* **where** *f0*: $f\ 0 = z$ **and** *seq*: $\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r$
by *iprover*
{
fix *i* **have** (($\lambda i.$ *case i of 0* $\Rightarrow$ *a* | *Suc i* $\Rightarrow$ *f i*) *i*, *f i*) $\in r$
using *seq a-z f0*
by (*cases i*) *auto*
}
then
show $\exists f.\ f\ 0 = a \wedge (\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r)$
by − (*rule exI* [**where** *x*=$\lambda i.$ *case i of 0* $\Rightarrow$ *a* | *Suc i* $\Rightarrow$ *f i*],*simp*)
qed

lemma *renumber*:
$\forall i.$ (a,f i) $\in r^*$ $\wedge$ (f i,f(*Suc* i)) $\in r$
$\Longrightarrow \exists f.\ f\ 0 = a \wedge (\forall i.\ (f\ i,\ f(Suc\ i)) \in r)$
by (*blast dest*:*renumber'*)

lemma *lem*:
$\forall y.\ r^{++}\ a\ y \longrightarrow P\ a \longrightarrow P\ y$
$\Longrightarrow ((b,a) \in \{(y,x).\ P\ x \wedge r\ x\ y\}^+) = ((b,a) \in \{(y,x).\ P\ x \wedge r^{++}\ x\ y\})$

```
apply(rule iffI)
 apply clarify
 apply(erule trancl-induct)
  apply blast
 apply(blast intro:tranclp-trans)
apply clarify
apply(erule tranclp-induct)
 apply blast
apply(blast intro:trancl-trans)
done

corollary terminates-impl-no-infinite-trans-computation:
 assumes terminates: Γ⊢c↓s
 shows ¬(∃f. f 0 = (c,s) ∧ (∀i. Γ⊢f i →⁺ f(Suc i)))
proof −
  have wf({(y,x). Γ⊢(c,s) →* x ∧ Γ⊢x → y}⁺)
  proof (rule wf-trancl)
    show wf {(y, x). Γ⊢(c,s) →* x ∧ Γ⊢x → y}
    proof (simp only: wf-iff-no-infinite-down-chain,clarify,simp)
      fix f
      assume ∀i. Γ⊢(c,s) →* f i ∧ Γ⊢f i → f (Suc i)
      hence ∃f. f (0::nat) = (c,s) ∧ (∀i. Γ⊢f i → f (Suc i))
        by (rule renumber [to-pred])
      moreover from terminates-impl-no-infinite-computation [OF terminates]
      have ¬ (∃f. f (0::nat) = (c, s) ∧ (∀i. Γ⊢f i → f (Suc i)))
        by (simp add: inf-def)
      ultimately show False
        by simp
    qed
  qed
  hence ¬ (∃f. ∀i. (f (Suc i), f i)
               ∈ {(y, x). Γ⊢(c, s) →* x ∧ Γ⊢x → y}⁺)
    by (simp add: wf-iff-no-infinite-down-chain)
  thus ?thesis
  proof (rule contrapos-nn)
    assume ∃f. f (0::nat) = (c, s) ∧ (∀i. Γ⊢f i →⁺ f (Suc i))
    then obtain f where
      f0: f 0 = (c, s) and
      seq: ∀i. Γ⊢f i →⁺ f (Suc i)
      by iprover
    show
      ∃f. ∀i. (f (Suc i), f i) ∈ {(y, x). Γ⊢(c, s) →* x ∧ Γ⊢x → y}⁺
    proof (rule exI [where x=f],rule allI)
      fix i
      show (f (Suc i), f i) ∈ {(y, x). Γ⊢(c, s) →* x ∧ Γ⊢x → y}⁺
      proof −
        {
          fix i have Γ⊢(c,s) →* f i
          proof (induct i)
```

case *0* **show** $\Gamma\vdash(c,\ s) \to^* f\ 0$
  by (*simp add*: *f0*)
next
  case (*Suc n*)
  have $\Gamma\vdash(c,\ s) \to^* f\ n$ **by** *fact*
  with *seq* **show** $\Gamma\vdash(c,\ s) \to^* f\ (Suc\ n)$
    by (*blast intro*: *tranclp-into-rtranclp rtranclp-trans*)
qed
}
hence $\Gamma\vdash(c,s) \to^* f\ i$
  by *iprover*
with *seq* **have**
  $(f\ (Suc\ i),\ f\ i) \in \{(y,\ x).\ \Gamma\vdash(c,\ s) \to^* x \wedge \Gamma\vdash x \to^+ y\}$
  by *clarsimp*
moreover
have $\forall y.\ \Gamma\vdash f\ i \to^+ y \longrightarrow \Gamma\vdash(c,\ s) \to^* f\ i \longrightarrow \Gamma\vdash(c,\ s) \to^* y$
  by (*blast intro*: *tranclp-into-rtranclp rtranclp-trans*)
ultimately
show *?thesis*
  by (*subst lem*)
qed
qed
qed
qed

theorem *wf-termi-call-steps*: *wf* (*termi-call-steps* Γ)
proof (*simp only*: *termi-call-steps-def wf-iff-no-infinite-down-chain*,
  clarify,*simp*)
  fix *f*
  assume *inf*: $\forall i.\ (\lambda(t,\ q)\ (s,\ p).$
      $\Gamma\vdash Call\ p \downarrow Normal\ s \wedge$
      $(\exists c.\ \Gamma\vdash\ (Call\ p,\ Normal\ s) \to^+ (c,\ Normal\ t) \wedge redex\ c = Call\ q))$
    $(f\ (Suc\ i))\ (f\ i)$
  define *s* **where** $s\ i = fst\ (f\ i)$ **for** $i{::}nat$
  define *p* **where** $p\ i = (snd\ (f\ i){::}'b)$ **for** $i :: nat$
  from *inf*
  have *inf'*: $\forall i.\ \Gamma\vdash Call\ (p\ i) \downarrow Normal\ (s\ i) \wedge$
      $(\exists c.\ \Gamma\vdash\ (Call\ (p\ i),\ Normal\ (s\ i)) \to^+ (c,\ Normal\ (s\ (i{+}1))) \wedge$
        $redex\ c = Call\ (p\ (i{+}1)))$
    apply −
    apply (*rule allI*)
    apply (*erule-tac x=i* **in** *allE*)
    apply (*auto simp add*: *s-def p-def*)
    done
  show *False*
  proof −
    from *inf'*
    have $\exists c.\ \forall i.\ \Gamma\vdash Call\ (p\ i) \downarrow Normal\ (s\ i) \wedge$
      $\Gamma\vdash\ (Call\ (p\ i),\ Normal\ (s\ i)) \to^+ (c\ i,\ Normal\ (s\ (i{+}1))) \wedge$

```
              redex (c i) = Call (p (i+1))
      apply -
      apply (rule choice)
      by blast
    then obtain c where
      termi-c: ∀ i. Γ⊢Call (p i) ↓ Normal (s i) and
      steps-c: ∀ i. Γ⊢ (Call (p i), Normal (s i)) →+ (c i, Normal (s (i+1))) and
      red-c:   ∀ i. redex (c i) = Call (p (i+1))
      by auto
    define g where g i = (seq c (p 0) i,Normal (s i)::('a,'c) xstate) for i
    from red-c [rule-format, of 0]
    have g 0 = (Call (p 0), Normal (s 0))
      by (simp add: g-def)
    moreover
    {
      fix i
      have redex (seq c (p 0) i) = Call (p i)
        by (induct i) (auto simp add: redex-subst-redex red-c)
      from this [symmetric]
      have subst-redex (seq c (p 0) i) (Call (p i)) = (seq c (p 0) i)
        by (simp add: subst-redex-redex)
    } note subst-redex-seq = this
    have ∀ i. Γ⊢ (g i) →+ (g (i+1))
    proof
      fix i
      from steps-c [rule-format, of i]
      have Γ⊢ (Call (p i), Normal (s i)) →+ (c i, Normal (s (i + 1))).
      from steps-redex' [OF this, of (seq c (p 0) i)]
      have Γ⊢ (subst-redex (seq c (p 0) i) (Call (p i)), Normal (s i)) →+
               (subst-redex (seq c (p 0) i) (c i), Normal (s (i + 1))) .
      hence Γ⊢ (seq c (p 0) i, Normal (s i)) →+
                (seq c (p 0) (i+1), Normal (s (i + 1)))
        by (simp add: subst-redex-seq)
      thus Γ⊢ (g i) →+ (g (i+1))
        by (simp add: g-def)
    qed
    moreover
    from terminates-impl-no-infinite-trans-computation [OF termi-c [rule-format,
of 0]]
    have ¬ (∃f. f 0 = (Call (p 0), Normal (s 0)) ∧ (∀ i. Γ⊢ f i →+ f (Suc i))) .
    ultimately show False
      by auto
  qed
qed


lemma no-infinite-computation-implies-wf:
  assumes not-inf: ¬ Γ⊢ (c, s) → ...(∞)
  shows wf {(c2,c1). Γ ⊢ (c,s) →* c1 ∧ Γ ⊢ c1 → c2}
```

proof (*simp only*: *wf-iff-no-infinite-down-chain*,*clarify*, *simp*)
  fix *f*
  assume $\forall i.\ \Gamma\vdash(c,\ s) \rightarrow^* f\ i \wedge \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
  hence $\exists f.\ f\ 0 = (c,\ s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i))$
    by (*rule renumber* [*to-pred*])
  moreover from *not-inf*
  have $\neg\ (\exists f.\ f\ 0 = (c,\ s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)))$
    by (*simp add*: *inf-def*)
  ultimately show *False*
    by *simp*
qed

lemma *not-final-Stuck-step*: $\neg$ *final* (*c*,*Stuck*) $\Longrightarrow \exists c'\ s'.\ \Gamma\vdash (c,\ Stuck) \rightarrow (c',s')$
by (*induct c*) (*fastforce intro*: *step.intros simp add*: *final-def*)+

lemma *not-final-Abrupt-step*:
  $\neg$ *final* (*c*,*Abrupt s*) $\Longrightarrow \exists c'\ s'.\ \Gamma\vdash (c,\ Abrupt\ s) \rightarrow (c',s')$
by (*induct c*) (*fastforce intro*: *step.intros simp add*: *final-def*)+

lemma *not-final-Fault-step*:
  $\neg$ *final* (*c*,*Fault f*) $\Longrightarrow \exists c'\ s'.\ \Gamma\vdash (c,\ Fault\ f) \rightarrow (c',s')$
by (*induct c*) (*fastforce intro*: *step.intros simp add*: *final-def*)+

lemma *not-final-Normal-step*:
  $\neg$ *final* (*c*,*Normal s*) $\Longrightarrow \exists c'\ s'.\ \Gamma\vdash (c,\ Normal\ s) \rightarrow (c',s')$
proof (*induct c*)
  case *Skip* **thus** *?case* **by** (*fastforce intro*: *step.intros simp add*: *final-def*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *step.intros*)
next
  case (*Spec r*)
  thus *?case*
    by (*cases* $\exists t.\ (s,t) \in r$) (*fastforce intro*: *step.intros*)+
next
  case (*Seq* c_1 c_2)
  thus *?case*
   by (*cases final* (c_1,*Normal s*)) (*fastforce intro*: *step.intros simp add*: *final-def*)+
next
  case (*Cond b c1 c2*)
  show *?case*
    by (*cases* $s \in b$) (*fastforce intro*: *step.intros*)+
next
  case (*While b c*)
  show *?case*
    by (*cases* $s \in b$) (*fastforce intro*: *step.intros*)+
next
  case (*Call p*)
  show *?case*
  by (*cases* Γ *p*) (*fastforce intro*: *step.intros*)+

next
  case *DynCom* **thus** *?case* **by** (*fastforce intro*: *step.intros*)
next
  case (*Guard f g c*)
  show *?case*
    by (*cases* $s \in g$) (*fastforce intro*: *step.intros*)+
next
  case *Throw*
  thus *?case* **by** (*fastforce intro*: *step.intros simp add*: *final-def*)
next
  case (*Catch* c_1 c_2)
  thus *?case*
    by (*cases final* (c_1,*Normal s*)) (*fastforce intro*: *step.intros simp add*: *final-def*)+
qed

lemma *final-termi*:
final $(c,s) \Longrightarrow \Gamma\vdash c\downarrow s$
  by (*cases s*) (*auto simp add*: *final-def terminates.intros*)

lemma *split-computation*:
assumes *steps*: $\Gamma\vdash (c,\ s) \rightarrow^* (c_f,\ s_f)$
assumes *not-final*: $\neg$ *final* (c,s)
assumes *final*: *final* (c_f,s_f)
shows $\exists\, c'\ s'.\ \Gamma\vdash (c,\ s) \rightarrow (c',s') \wedge \Gamma\vdash (c',\ s') \rightarrow^* (c_f,\ s_f)$
using *steps not-final final*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl* **thus** *?case* **by** *simp*
next
  case (*Trans* $c\ s\ c'\ s'$)
  thus *?case* **by** *auto*
qed

lemma *wf-implies-termi-reach-step-case*:
assumes *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash (c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$
shows $\Gamma\vdash c \downarrow Normal\ s$
using *hyp*
proof (*induct c*)
  case *Skip* **show** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Basic* **show** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Spec r*)
  show *?case*
    by (*cases* $\exists\, t.\ (s,t)\in r$) (*fastforce intro*: *terminates.intros*)+
next
  case (*Seq* c_1 c_2)
  have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash (Seq\ c_1\ c_2,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  show *?case*

proof (*rule terminates.Seq*)
  {
    fix c' s'
    assume *step-c*$_1$: $\Gamma\vdash$ (c_1, *Normal* s) $\rightarrow$ (c', s')
    have $\Gamma\vdash c' \downarrow s'$
    proof −
      from *step-c*$_1$
      have $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow$ (*Seq* c' c_2, s')
        by (*rule step.Seq*)
      from *hyp* [*OF this*]
      have $\Gamma\vdash$*Seq* c' $c_2 \downarrow s'$.
      thus $\Gamma\vdash c'\downarrow s'$
        by *cases auto*
    qed
  }
  from *Seq.hyps* (*1*) [*OF this*]
  show $\Gamma\vdash c_1 \downarrow$ *Normal* s.
next
  show $\forall s'$. $\Gamma\vdash$ $\langle c_1$,*Normal* $s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash c_2 \downarrow s'$
  proof (*intro allI impI*)
    fix s'
    assume *exec-c*$_1$: $\Gamma\vdash$ $\langle c_1$,*Normal* $s\rangle \Rightarrow s'$
    show $\Gamma\vdash c_2 \downarrow s'$
    proof (*cases final* (c_1,*Normal* s))
      case *True*
      hence c_1=*Skip* $\vee$ c_1=*Throw*
        by (*simp add*: *final-def*)
      thus *?thesis*
      proof
        assume *Skip*: c_1=*Skip*
        have $\Gamma\vdash$(*Seq Skip* c_2,*Normal* s) $\rightarrow$ (c_2,*Normal* s)
          by (*rule step.SeqSkip*)
        from *hyp* [*simplified Skip*, *OF this*]
        have $\Gamma\vdash c_2 \downarrow$ *Normal* s .
        moreover from *exec-c*$_1$ *Skip*
        have s'=*Normal* s
          by (*auto elim*: *exec-Normal-elim-cases*)
        ultimately show *?thesis* **by** *simp*
      next
        assume *Throw*: c_1=*Throw*
        with *exec-c*$_1$ **have** s'=*Abrupt* s
          by (*auto elim*: *exec-Normal-elim-cases*)
        thus *?thesis*
          by *auto*
      qed
    next
      case *False*
      from *exec-impl-steps* [*OF exec-c*$_1$]
      obtain c_f t **where**

steps-c_1: $\Gamma\vdash$ (c_1, *Normal* s) $\rightarrow^*$ (c_f, t) **and**
fin:(*case* s' *of*
Abrupt $x \Rightarrow c_f$ = *Throw* $\wedge$ t = *Normal* x
| - $\Rightarrow c_f$ = *Skip* $\wedge$ $t = s'$)
by (*fastforce split*: *xstate.splits*)
with *fin* **have** *final*: *final* (c_f,t)
by (*cases* s') (*auto simp add*: *final-def*)
from *split-computation* [*OF steps-*c_1 *False this*]
obtain c'' s'' **where**
first: $\Gamma\vdash$ (c_1, *Normal* s) $\rightarrow$ (c'', s'') **and**
rest: $\Gamma\vdash$ (c'', s'') $\rightarrow^*$ (c_f, t)
by *blast*
from *step.Seq* [*OF first*]
have $\Gamma\vdash$ (*Seq* c_1 c_2, *Normal* s) $\rightarrow$ (*Seq* c'' c_2, s'')**.**
from *hyp* [*OF this*]
have *termi-*s'': $\Gamma\vdash$*Seq* c'' $c_2 \downarrow s''$**.**
show *?thesis*
proof (*cases* s'')
case (*Normal* x)
from *termi-*s'' [*simplified Normal*]
have *termi-*c_2: $\forall t.\ \Gamma\vdash \langle c''$,*Normal* $x\rangle \Rightarrow t \longrightarrow \Gamma\vdash c_2 \downarrow t$
by *cases*
show *?thesis*
proof (*cases* $\exists x'.\ s'$=*Abrupt* x')
case *False*
with *fin* **obtain** c_f=*Skip* $t=s'$
by (*cases* s') *auto*
from *steps-Skip-impl-exec* [*OF rest* [*simplified this*]] *Normal*
have $\Gamma\vdash \langle c''$,*Normal* $x\rangle \Rightarrow s'$
by *simp*
from *termi-*c_2 [*rule-format*, *OF this*]
show $\Gamma\vdash c_2 \downarrow s'$ **.**
next
case *True*
with *fin* **obtain** x' **where** s': s'=*Abrupt* x' **and** c_f=*Throw* t=*Normal*
x'
by *auto*
from *steps-Throw-impl-exec* [*OF rest* [*simplified this*]] *Normal*
have $\Gamma\vdash \langle c''$,*Normal* $x\rangle \Rightarrow$ *Abrupt* x'
by *simp*
from *termi-*c_2 [*rule-format*, *OF this*] s'
show $\Gamma\vdash c_2 \downarrow s'$ **by** *simp*
qed
next
case (*Abrupt* x)
from *steps-Abrupt-prop* [*OF rest this*]
have t=*Abrupt* x **by** *simp*
with *fin* **have** s'=*Abrupt* x
by (*cases* s') *auto*

```
        thus Γ⊢c₂ ↓ s'
          by auto
      next
        case (Fault f)
        from steps-Fault-prop [OF rest this]
        have t=Fault f by simp
        with fin have s'=Fault f
          by (cases s') auto
        thus Γ⊢c₂ ↓ s'
          by auto
      next
        case Stuck
        from steps-Stuck-prop [OF rest this]
        have t=Stuck by simp
        with fin have s'=Stuck
          by (cases s') auto
        thus Γ⊢c₂ ↓ s'
          by auto
      qed
    qed
  qed
 qed
next
 case (Cond b c₁ c₂)
 have hyp: ⋀c' s'. Γ⊢ (Cond b c₁ c₂, Normal s) → (c', s') ⟹ Γ⊢c' ↓ s' by fact
 show ?case
 proof (cases s∈b)
   case True
   then have Γ⊢ (Cond b c₁ c₂, Normal s) → (c₁, Normal s)
     by (rule step.CondTrue)
   from hyp [OF this] have Γ⊢c₁ ↓ Normal s.
   with True show ?thesis
     by (auto intro: terminates.intros)
 next
   case False
   then have Γ⊢ (Cond b c₁ c₂, Normal s) → (c₂, Normal s)
     by (rule step.CondFalse)
   from hyp [OF this] have Γ⊢c₂ ↓ Normal s.
   with False show ?thesis
     by (auto intro: terminates.intros)
 qed
next
 case (While b c)
 have hyp: ⋀c' s'. Γ⊢ (While b c, Normal s) → (c', s') ⟹ Γ⊢c' ↓ s' by fact
 show ?case
 proof (cases s∈b)
   case True
   then have Γ⊢ (While b c, Normal s) → (Seq c (While b c), Normal s)
     by (rule step.WhileTrue)
```

from *hyp* [*OF this*] **have** $\Gamma\vdash(Seq\ c\ (While\ b\ c)) \downarrow Normal\ s$**.**
with *True* **show** *?thesis*
by (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*)
next
case *False*
thus *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*Call p*)
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Call\ p,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof (*cases* Γ *p*)
case *None*
thus *?thesis*
by (*auto intro*: *terminates.intros*)
next
case (*Some bdy*)
then have $\Gamma\vdash\ (Call\ p,\ Normal\ s) \rightarrow (bdy,\ Normal\ s)$
by (*rule step.Call*)
from *hyp* [*OF this*] **have** $\Gamma\vdash bdy \downarrow Normal\ s$**.**
with *Some* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*DynCom c*)
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (DynCom\ c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
have $\Gamma\vdash\ (DynCom\ c,\ Normal\ s) \rightarrow (c\ s,\ Normal\ s)$
by (*rule step.DynCom*)
from *hyp* [*OF this*] **have** $\Gamma\vdash c\ s \downarrow Normal\ s$**.**
then show *?case*
by (*auto intro*: *terminates.intros*)
next
case (*Guard f g c*)
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Guard\ f\ g\ c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof (*cases* $s{\in}g$)
case *True*
then have $\Gamma\vdash\ (Guard\ f\ g\ c,\ Normal\ s) \rightarrow (c,\ Normal\ s)$
by (*rule step.Guard*)
from *hyp* [*OF this*] **have** $\Gamma\vdash c{\downarrow}\ Normal\ s$**.**
with *True* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
next
case *False*
thus *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next

case *Throw* **show** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Catch* c_1 c_2)
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Catch\ c_1\ c_2,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof (*rule terminates.Catch*)
{
fix c' s'
assume *step-*c_1: $\Gamma\vdash\ (c_1,\ Normal\ s) \rightarrow (c',\ s')$
have $\Gamma\vdash c' \downarrow s'$
proof −
from *step-*c_1
have $\Gamma\vdash\ (Catch\ c_1\ c_2,\ Normal\ s) \rightarrow (Catch\ c'\ c_2,\ s')$
by (*rule step.Catch*)
from *hyp* [*OF this*]
have $\Gamma\vdash Catch\ c'\ c_2 \downarrow s'$**.**
thus $\Gamma\vdash c'\downarrow s'$
by *cases auto*
qed
}
from *Catch.hyps* (*1*) [*OF this*]
show $\Gamma\vdash c_1 \downarrow Normal\ s$**.**
next
show $\forall s'.\ \Gamma\vdash\ \langle c_1,Normal\ s\rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash c_2 \downarrow Normal\ s'$
proof (*intro allI impI*)
fix s'
assume *exec-*c_1: $\Gamma\vdash\ \langle c_1,Normal\ s\rangle \Rightarrow Abrupt\ s'$
show $\Gamma\vdash c_2 \downarrow Normal\ s'$
proof (*cases final* (c_1,*Normal s*))
case *True*
with *exec-*c_1
have *Throw*: c_1=*Throw*
by (*auto simp add*: *final-def elim*: *exec-Normal-elim-cases*)
have $\Gamma\vdash(Catch\ Throw\ c_2,Normal\ s) \rightarrow (c_2,Normal\ s)$
by (*rule step.CatchThrow*)
from *hyp* [*simplified Throw*, *OF this*]
have $\Gamma\vdash c_2 \downarrow Normal\ s$**.**
moreover from *exec-*c_1 *Throw*
have s'=s
by (*auto elim*: *exec-Normal-elim-cases*)
ultimately show *?thesis* **by** *simp*
next
case *False*
from *exec-impl-steps* [*OF exec-*c_1]
obtain c_f t **where**
steps-c_1: $\Gamma\vdash\ (c_1,\ Normal\ s) \rightarrow^* (Throw,\ Normal\ s')$
by (*fastforce split*: *xstate.splits*)
from *split-computation* [*OF steps-*c_1 *False*]
obtain c'' s'' **where**

```
      first: Γ⊢ (c₁, Normal s) → (c'', s'') and
      rest: Γ⊢ (c'', s'') →* (Throw, Normal s')
      by (auto simp add: final-def)
    from step.Catch [OF first]
    have Γ⊢ (Catch c₁ c₂, Normal s) → (Catch c'' c₂, s'').
    from hyp [OF this]
    have Γ⊢Catch c'' c₂ ↓ s''.
    moreover
    from steps-Throw-impl-exec [OF rest]
    have Γ⊢ ⟨c'',s''⟩ ⇒ Abrupt s'.
    moreover
    from rest obtain x where s''=Normal x
      by (cases s'')
         (auto dest: steps-Fault-prop steps-Abrupt-prop steps-Stuck-prop)
    ultimately show ?thesis
      by (fastforce elim: terminates-elim-cases)
   qed
  qed
 qed
qed

lemma wf-implies-termi-reach:
assumes wf: wf {(cfg2,cfg1). Γ ⊢ (c,s) →* cfg1 ∧ Γ ⊢ cfg1 → cfg2}
shows ⋀c1 s1. ⟦Γ ⊢ (c,s) →* cfg1;  cfg1=(c1,s1)⟧⟹ Γ⊢c1↓s1
using wf
proof (induct cfg1, simp)
 fix c1 s1
 assume reach: Γ⊢ (c, s) →* (c1, s1)
 assume hyp-raw: ⋀y c2 s2.
        ⟦Γ⊢ (c1, s1) → (c2, s2); Γ⊢ (c, s) →* (c2, s2); y = (c2, s2)⟧
        ⟹ Γ⊢c2 ↓ s2
 have hyp: ⋀c2 s2. Γ⊢ (c1, s1) → (c2, s2) ⟹ Γ⊢c2 ↓ s2
  apply −
  apply (rule hyp-raw)
  apply  assumption
  using reach
  apply  simp
  apply (rule refl)
  done

 show Γ⊢c1 ↓ s1
 proof (cases s1)
  case (Normal s1')
  with wf-implies-termi-reach-step-case [OF hyp [simplified Normal]]
  show ?thesis
   by auto
 qed (auto intro: terminates.intros)
qed
```

theorem *no-infinite-computation-impl-terminates*:
  assumes *not-inf*: $\neg\ \Gamma\vdash\ (c,\ s) \to \ldots(\infty)$
  shows $\Gamma\vdash c\downarrow s$
proof −
  from *no-infinite-computation-implies-wf* [*OF not-inf*]
  have *wf*: $wf\ \{(c2,\ c1).\ \Gamma\vdash(c,\ s) \to^*\ c1 \wedge \Gamma\vdash c1 \to c2\}$.
  show *?thesis*
    by (*rule wf-implies-termi-reach* [*OF wf*]) *auto*
qed

corollary *terminates-iff-no-infinite-computation*:
  $\Gamma\vdash c\downarrow s = (\neg\ \Gamma\vdash\ (c,\ s) \to \ldots(\infty))$
  apply (*rule*)
  apply (*erule terminates-impl-no-infinite-computation*)
  apply (*erule no-infinite-computation-impl-terminates*)
  done

4.6 Generalised Redexes

For an important lemma for the completeness proof of the Hoare-logic for total correctness we need a generalisation of *redex* that not only yield the redex itself but all the enclosing statements as well.

primrec *redexes*:: $('s,'p,'f)com \Rightarrow ('s,'p,'f)com\ set$
where
$redexes\ Skip = \{Skip\}$ |
$redexes\ (Basic\ f) = \{Basic\ f\}$ |
$redexes\ (Spec\ r) = \{Spec\ r\}$ |
$redexes\ (Seq\ c_1\ c_2) = \{Seq\ c_1\ c_2\} \cup redexes\ c_1$ |
$redexes\ (Cond\ b\ c_1\ c_2) = \{Cond\ b\ c_1\ c_2\}$ |
$redexes\ (While\ b\ c) = \{While\ b\ c\}$ |
$redexes\ (Call\ p) = \{Call\ p\}$ |
$redexes\ (DynCom\ d) = \{DynCom\ d\}$ |
$redexes\ (Guard\ f\ b\ c) = \{Guard\ f\ b\ c\}$ |
$redexes\ (Throw) = \{Throw\}$ |
$redexes\ (Catch\ c_1\ c_2) = \{Catch\ c_1\ c_2\} \cup redexes\ c_1$

lemma *root-in-redexes*: $c \in redexes\ c$
  apply (*induct c*)
  apply *auto*
  done

lemma *redex-in-redexes*: $redex\ c \in redexes\ c$
  apply (*induct c*)
  apply *auto*
  done

lemma *redex-redexes*: $\bigwedge c'.\ [\![c' \in redexes\ c;\ redex\ c' = c']\!] \Longrightarrow redex\ c = c'$
  apply (*induct c*)
  apply *auto*

done

lemma *step-redexes*:
shows $\bigwedge r\ r'.\ [\![\Gamma\vdash(r,s) \rightarrow (r',s');\ r \in \mathit{redexes}\ c]\!]$
$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \rightarrow (c',s') \wedge r' \in \mathit{redexes}\ c'$
proof (*induct c*)
case *Skip* **thus** *?case* **by** (*fastforce intro*: *step.intros elim*: *step-elim-cases*)
next
case *Basic* **thus** *?case* **by** (*fastforce intro*: *step.intros elim*: *step-elim-cases*)
next
case *Spec* **thus** *?case* **by** (*fastforce intro*: *step.intros elim*: *step-elim-cases*)
next
case (*Seq* c_1 c_2)
have $r \in \mathit{redexes}\ (\mathit{Seq}\ c_1\ c_2)$ **by** *fact*
hence r: $r = \mathit{Seq}\ c_1\ c_2 \vee r \in \mathit{redexes}\ c_1$
by *simp*
have *step-r*: $\Gamma\vdash (r,\ s) \rightarrow (r',\ s')$ **by** *fact*
from r **show** *?case*
proof
assume $r = \mathit{Seq}\ c_1\ c_2$
with *step-r*
show *?case*
by (*auto simp add*: *root-in-redexes*)
next
assume r: $r \in \mathit{redexes}\ c_1$
from *Seq.hyps* (*1*) [*OF step-r this*]
obtain c' **where**
step-c_1: $\Gamma\vdash (c_1,\ s) \rightarrow (c',\ s')$ **and**
r': $r' \in \mathit{redexes}\ c'$
by *blast*
from *step.Seq* [*OF step-*c_1]
have $\Gamma\vdash (\mathit{Seq}\ c_1\ c_2,\ s) \rightarrow (\mathit{Seq}\ c'\ c_2,\ s')$.
with r'
show *?case*
by *auto*
qed
next
case *Cond*
thus *?case*
by (*fastforce intro*: *step.intros elim*: *step-elim-cases simp add*: *root-in-redexes*)
next
case *While*
thus *?case*
by (*fastforce intro*: *step.intros elim*: *step-elim-cases simp add*: *root-in-redexes*)
next
case *Call* **thus** *?case*
by (*fastforce intro*: *step.intros elim*: *step-elim-cases simp add*: *root-in-redexes*)
next
case *DynCom* **thus** *?case*

```
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case Guard thus ?case
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case Throw thus ?case
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case (Catch c1 c2)
  have r ∈ redexes (Catch c1 c2) by fact
  hence r: r = Catch c1 c2 ∨ r ∈ redexes c1
    by simp
  have step-r: Γ⊢ (r, s) → (r', s') by fact
  from r show ?case
  proof
    assume r = Catch c1 c2
    with step-r
    show ?case
      by (auto simp add: root-in-redexes)
  next
    assume r: r ∈ redexes c1
    from Catch.hyps (1) [OF step-r this]
    obtain c' where
      step-c1: Γ⊢ (c1, s) → (c', s') and
      r': r' ∈ redexes c'
      by blast
    from step.Catch [OF step-c1]
    have Γ⊢ (Catch c1 c2, s) → (Catch c' c2, s').
    with r'
    show ?case
      by auto
  qed
qed

lemma steps-redexes:
  assumes steps: Γ⊢ (r, s) →* (r', s')
  shows ⋀c. r ∈ redexes c ⟹ ∃c'. Γ⊢(c,s) →* (c',s') ∧ r' ∈ redexes c'
using steps
proof (induct rule: converse-rtranclp-induct2 [case-names Refl Trans])
  case Refl
  then
  show ∃c'. Γ⊢ (c, s') →* (c', s') ∧ r' ∈ redexes c'
    by auto
next
  case (Trans r s r'' s'')
  have Γ⊢ (r, s) → (r'', s'') r ∈ redexes c by fact+
  from step-redexes [OF this]
  obtain c' where
    step: Γ⊢ (c, s) → (c', s'') and
```

r'': $r'' \in$ *redexes* c'
by *blast*
note *step*
also
from *Trans.hyps* (*3*) [*OF* r']
obtain c'' **where**
steps: $\Gamma\vdash (c', s'') \rightarrow^* (c'', s')$ **and**
r': $r' \in$ *redexes* c''
by *blast*
note *steps*
finally
show *?case*
using r'
by *blast*
qed

lemma *steps-redexes'*:
assumes *steps*: $\Gamma\vdash (r, s) \rightarrow^+ (r', s')$
shows $\bigwedge c.\ r \in$ *redexes* $c \Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \rightarrow^+ (c',s') \wedge r' \in$ *redexes* c'
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*, *case-names Step Trans*])
case (*Step* r' s' c')
have $\Gamma\vdash (r, s) \rightarrow (r', s')$ $r \in$ *redexes* c' **by** *fact+*
from *step-redexes* [*OF this*]
show *?case*
by (*blast intro*: *r-into-trancl*)
next
case (*Trans* r' s' r'' s'')
from *Trans* **obtain** c' **where**
steps: $\Gamma\vdash (c, s) \rightarrow^+ (c', s')$ **and**
r': $r' \in$ *redexes* c'
by *blast*
note *steps*
moreover
have $\Gamma\vdash (r', s') \rightarrow (r'', s'')$ **by** *fact*
from *step-redexes* [*OF this* r'] **obtain** c'' **where**
step: $\Gamma\vdash (c', s') \rightarrow (c'', s'')$ **and**
r'': $r'' \in$ *redexes* c''
by *blast*
note *step*
finally show *?case*
using r'' **by** *blast*
qed

lemma *step-redexes-Seq*:
assumes *step*: $\Gamma\vdash(r,s) \rightarrow (r',s')$
assumes *Seq*: *Seq* r $c_2 \in$ *redexes* c

shows $\exists\, c'.\ \Gamma\vdash(c,s) \to (c',s') \wedge$ *Seq r' c*$_2$ $\in$ *redexes c'*
proof $-$
from *step.Seq* [*OF step*]
have $\Gamma\vdash$ (*Seq r c*$_2$, *s*) $\to$ (*Seq r' c*$_2$, *s'*).
from *step-redexes* [*OF this Seq*]
show *?thesis* .
qed

lemma *steps-redexes-Seq*:
assumes *steps*: $\Gamma\vdash (r,\ s) \to^* (r',\ s')$
shows $\bigwedge c.$ *Seq r c*$_2$ $\in$ *redexes c* $\Longrightarrow$
$\exists\, c'.\ \Gamma\vdash(c,s) \to^* (c',s') \wedge$ *Seq r' c*$_2$ $\in$ *redexes c'*
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl*
then show *?case*
by (*auto*)

next
case (*Trans r s r'' s''*)
have $\Gamma\vdash (r,\ s) \to (r'',\ s'')$ *Seq r c*$_2$ $\in$ *redexes c* **by** *fact+*
from *step-redexes-Seq* [*OF this*]
obtain *c'* **where**
step: $\Gamma\vdash (c,\ s) \to (c',\ s'')$ **and**
r'': *Seq r'' c*$_2$ $\in$ *redexes c'*
by *blast*
note *step*
also
from *Trans.hyps* (*3*) [*OF r''*]
obtain *c''* **where**
steps: $\Gamma\vdash (c',\ s'') \to^* (c'',\ s')$ **and**
r': *Seq r' c*$_2$ $\in$ *redexes c''*
by *blast*
note *steps*
finally
show *?case*
using *r'*
by *blast*
qed

lemma *steps-redexes-Seq'*:
assumes *steps*: $\Gamma\vdash (r,\ s) \to^+ (r',\ s')$
shows $\bigwedge c.$ *Seq r c*$_2$ $\in$ *redexes c*
$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \to^+ (c',s') \wedge$ *Seq r' c*$_2$ $\in$ *redexes c'*
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*, *case-names Step Trans*])
case (*Step r' s' c'*)
have $\Gamma\vdash (r,\ s) \to (r',\ s')$ *Seq r c*$_2$ $\in$ *redexes c'* **by** *fact+*
from *step-redexes-Seq* [*OF this*]

show *?case*
by (*blast intro*: *r-into-trancl*)
next
case (*Trans* r' s' r'' s'')
from *Trans* **obtain** c' **where**
steps: $\Gamma\vdash (c,\ s) \rightarrow^+ (c',\ s')$ **and**
r': *Seq* r' $c_2 \in$ *redexes* c'
by *blast*
note *steps*
moreover
have $\Gamma\vdash (r',\ s') \rightarrow (r'',\ s'')$ **by** *fact*
from *step-redexes-Seq* [*OF this* r'] **obtain** c'' **where**
step: $\Gamma\vdash (c',\ s') \rightarrow (c'',\ s'')$ **and**
r'': *Seq* r'' $c_2 \in$ *redexes* c''
by *blast*
note *step*
finally show *?case*
using r'' **by** *blast*
qed

lemma *step-redexes-Catch*:
assumes *step*: $\Gamma\vdash(r,s) \rightarrow (r',s')$
assumes *Catch*: *Catch* r $c_2 \in$ *redexes* c
shows $\exists\, c'.\ \Gamma\vdash(c,s) \rightarrow (c',s') \wedge$ *Catch* r' $c_2 \in$ *redexes* c'
proof −
from *step.Catch* [*OF step*]
have $\Gamma\vdash$ (*Catch* r c_2, s) $\rightarrow$ (*Catch* r' c_2, s')**.**
from *step-redexes* [*OF this Catch*]
show *?thesis* **.**
qed

lemma *steps-redexes-Catch*:
assumes *steps*: $\Gamma\vdash (r,\ s) \rightarrow^* (r',\ s')$
shows $\bigwedge c.$ *Catch* r $c_2 \in$ *redexes* $c \Longrightarrow$
$\exists\, c'.\ \Gamma\vdash(c,s) \rightarrow^* (c',s') \wedge$ *Catch* r' $c_2 \in$ *redexes* c'
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl*
then show *?case*
by (*auto*)

next
case (*Trans* r s r'' s'')
have $\Gamma\vdash (r,\ s) \rightarrow (r'',\ s'')$ *Catch* r $c_2 \in$ *redexes* c **by** *fact+*
from *step-redexes-Catch* [*OF this*]
obtain c' **where**
step: $\Gamma\vdash (c,\ s) \rightarrow (c',\ s'')$ **and**
r'': *Catch* r'' $c_2 \in$ *redexes* c'
by *blast*

note *step*
also
from *Trans.hyps* (*3*) [*OF* r'']
obtain c'' **where**
steps: $\Gamma\vdash (c', s'') \rightarrow^* (c'', s')$ **and**
r': *Catch* r' $c_2 \in$ *redexes* c''
by *blast*
note *steps*
finally
show *?case*
using r'
by *blast*
qed

lemma *steps-redexes-Catch*$'$:
assumes *steps*: $\Gamma\vdash (r, s) \rightarrow^+ (r', s')$
shows $\bigwedge c.$ *Catch* r $c_2 \in$ *redexes* c
$\Longrightarrow \exists c'. \Gamma\vdash(c,s) \rightarrow^+ (c',s') \wedge$ *Catch* r' $c_2 \in$ *redexes* c'
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*, *case-names Step Trans*])
case (*Step* r' s' c')
have $\Gamma\vdash (r, s) \rightarrow (r', s')$ *Catch* r $c_2 \in$ *redexes* c' **by** *fact+*
from *step-redexes-Catch* [*OF this*]
show *?case*
by (*blast intro*: *r-into-trancl*)
next
case (*Trans* r' s' r'' s'')
from *Trans* **obtain** c' **where**
steps: $\Gamma\vdash (c, s) \rightarrow^+ (c', s')$ **and**
r': *Catch* r' $c_2 \in$ *redexes* c'
by *blast*
note *steps*
moreover
have $\Gamma\vdash (r', s') \rightarrow (r'', s'')$ **by** *fact*
from *step-redexes-Catch* [*OF this* r'] **obtain** c'' **where**
step: $\Gamma\vdash (c', s') \rightarrow (c'', s'')$ **and**
r'': *Catch* r'' $c_2 \in$ *redexes* c''
by *blast*
note *step*
finally show *?case*
using r'' **by** *blast*
qed

lemma *redexes-subset*: $\bigwedge c'. c' \in$ *redexes* $c \Longrightarrow$ *redexes* $c' \subseteq$ *redexes* c
by (*induct* c) *auto*

lemma *redexes-preserves-termination*:
assumes *termi*: $\Gamma\vdash c\downarrow s$
shows $\bigwedge c'. c' \in$ *redexes* $c \Longrightarrow \Gamma\vdash c'\downarrow s$

using *termi*
by *induct* (*auto intro*: *terminates.intros*)

end

5 The Simpl Syntax

theory *LanguageCon* **imports** *HOL−Library.Old-Recdef EmbSimpl/Language* **begin**

5.1 The Core Language

We use a shallow embedding of boolean expressions as well as assertions as sets of states.

type-synonym *'s bexp* = *'s set*
type-synonym *'s assn* = *'s set*

datatype (*dead 's*, *'p*, *'f*, *dead 'e*) *com* =
  Skip
 | *Basic 's* ⇒ *'s 'e option*
 | *Spec* (*'s* × *'s*) *set 'e option*
 | *Seq* (*'s* ,*'p*, *'f*,*'e*) *com* (*'s*,*'p*, *'f*,*'e*) *com*
 | *Cond 's bexp* (*'s*,*'p*,*'f*,*'e*) *com* (*'s*,*'p*,*'f*,*'e*) *com*
 | *While 's bexp* (*'s*,*'p*,*'f*,*'e*) *com*
 | *Call 'p*
 | *DynCom 's* ⇒ (*'s*,*'p*,*'f*,*'e*) *com*
 | *Guard 'f 's bexp* (*'s*,*'p*,*'f*,*'e*) *com*
 | *Throw*
 | *Catch* (*'s*,*'p*,*'f*,*'e*) *com* (*'s*,*'p*,*'f*,*'e*) *com*
 | *Await 's bexp* (*'s*,*'p*,*'f*) *Language.com 'e option*

primrec *sequential*:: (*'s*, *'p*, *'f*, *'e*) *com* ⇒ (*'s*,*'p*,*'f*) *Language.com*
where
sequential Skip = *Language.Skip* |
sequential (*Basic f e*) = *Language.Basic f* |
sequential (*Spec r e*) = *Language.Spec r* |
sequential (*Seq* c_1 c_2) = *Language.Seq* (*sequential* c_1) (*sequential* c_2) |
sequential (*Cond b* c_1 c_2) = *Language.Cond b* (*sequential* c_1) (*sequential* c_2) |
sequential (*While b c*) = *Language.While b* (*sequential c*) |
sequential (*Call p*) = *Language.Call p* |
sequential (*DynCom c*) = *Language.DynCom* (λ*s*. (*sequential* (*c s*))) |
sequential (*Guard f g c*) = *Language.Guard f g* (*sequential c*) |
sequential Throw = *Language.Throw* |
sequential (*Catch* c_1 c_2) = *Language.Catch* (*sequential* c_1) (*sequential* c_2) |
sequential (*Await b ca e*) = *Language.Skip*

primrec *parallel*:: $('s,'p,'f)$ *Language.com* $\Rightarrow$ $('s,'p,'f,'e)$ *com*
where
parallel Language.Skip = *Skip* |
parallel (*Language.Basic f*) = *Basic f None* |
parallel (*Language.Spec r*) = *Spec r None* |
parallel (*Language.Seq* c_1 c_2) = *Seq* (*parallel* c_1) (*parallel* c_2) |
parallel (*Language.Cond b* c_1 c_2) = *Cond b* (*parallel* c_1) (*parallel* c_2) |
parallel (*Language.While b c*) = *While b* (*parallel c*) |
parallel (*Language.Call p*) = *Call p* |
parallel (*Language.DynCom c*) = *DynCom* ($\lambda s.$ (*parallel* (*c s*))) |
parallel (*Language.Guard f g c*) = *Guard f g* (*parallel c*) |
parallel Language.Throw = *Throw* |
parallel (*Language.Catch* c_1 c_2) = *Catch* (*parallel* c_1) (*parallel* c_2)

primrec *noawaits*:: $('s, 'p, 'f, 'e)$ *com* $\Rightarrow$ *bool*
where
noawaits Skip = *True* |
noawaits (*Basic f e*) = *True* |
noawaits (*Spec r e*) = *True* |
noawaits (*Seq* c_1 c_2) = (*noawaits* c_1 $\wedge$ *noawaits* c_2) |
noawaits (*Cond b* c_1 c_2) = (*noawaits* c_1 $\wedge$ *noawaits* c_2) |
noawaits (*While b c*) = (*noawaits c*) |
noawaits (*Call p*) = *True* |
noawaits (*DynCom c*) = ($\forall s.$ *noawaits* (*c s*)) |
noawaits (*Guard f g c*) = *noawaits c* |
noawaits Throw = *True* |
noawaits (*Catch* c_1 c_2) = (*noawaits* c_1 $\wedge$ *noawaits* c_2) |
noawaits (*Await b cn e*) = *False*

5.2 Derived Language Constructs

definition
raise:: $('s \Rightarrow 's) \Rightarrow 'e$ *option* $\Rightarrow ('s, 'p, 'f, 'e)$ *com* **where**
raise f e = *Seq* (*Basic f e*) *Throw*

definition
condCatch:: $('s, 'p, 'f, 'e)$ *com* $\Rightarrow 's$ *bexp* $\Rightarrow ('s, 'p, 'f, 'e)$ *com* $\Rightarrow ('s, 'p, 'f, 'e)$ *com* **where**
condCatch c_1 *b* c_2 = *Catch* c_1 (*Cond b* c_2 *Throw*)

definition
bind:: $('s \Rightarrow 'v) \Rightarrow ('v \Rightarrow ('s, 'p, 'f, 'e)$ *com*$) \Rightarrow ('s, 'p, 'f, 'e)$ *com* **where**
bind e c = *DynCom* ($\lambda s.$ *c* (*e s*))

definition
bseq:: $('s, 'p, 'f, 'e)$ *com* $\Rightarrow ('s, 'p, 'f, 'e)$ *com* $\Rightarrow ('s, 'p, 'f, 'e)$ *com* **where**
bseq = *Seq*

definition
$block$:: $['s \Rightarrow 's, 'e\ option,\ ('s,\ 'p,\ 'f,\ 'e)\ com,\ 's \Rightarrow 's \Rightarrow 's,\ 'e\ option,\ 's \Rightarrow 's \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com] \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$
where
$block\ init\ ei\ bdy\ return\ er\ c =$
$DynCom\ (\lambda s.\ (Seq\ (Catch\ (Seq\ (Basic\ init\ ei)\ bdy)\ (Seq\ (Basic\ (return\ s)\ er)\ Throw))$
$(DynCom\ (\lambda t.\ Seq\ (Basic\ (return\ s)\ er)\ (c\ s\ t))))$
$)$

definition
$call$:: $('s \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow 'p \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow ('s \Rightarrow 's \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com) \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$ **where**
$call\ init\ ei\ p\ return\ er\ c = block\ init\ ei\ (Call\ p)\ return\ er\ c$

definition
$dynCall$:: $('s \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow ('s \Rightarrow 'p) \Rightarrow$
$('s \Rightarrow 's \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow ('s \Rightarrow 's \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com) \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$ **where**
$dynCall\ init\ ei\ p\ return\ er\ c = DynCom\ (\lambda s.\ call\ init\ ei\ (p\ s)\ return\ er\ c)$

definition
$fcall$:: $('s \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow 'p \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow 'e\ option \Rightarrow ('s \Rightarrow 'v) \Rightarrow ('v \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com)$
$\Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$ **where**
$fcall\ init\ ei\ p\ return\ er\ result\ c = call\ init\ ei\ p\ return\ er\ (\lambda s\ t.\ c\ (result\ t))$

definition
lem:: $'x \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$ **where**
$lem\ x\ c = c$

primrec $switch$:: $('s \Rightarrow 'v) \Rightarrow ('v\ set \times ('s,\ 'p,\ 'f,\ 'e)\ com)\ list \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$
where
$switch\ v\ [] = Skip\ |$
$switch\ v\ (Vc\#vs) = Cond\ \{s.\ v\ s \in fst\ Vc\}\ (snd\ Vc)\ (switch\ v\ vs)$

definition $guaranteeStrip$:: $'f \Rightarrow 's\ set \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$
where $guaranteeStrip\ f\ g\ c = Guard\ f\ g\ c$

definition $guaranteeStripPair$:: $'f \Rightarrow 's\ set \Rightarrow ('f \times 's\ set)$
where $guaranteeStripPair\ f\ g = (f,g)$

primrec $guards$:: $('f \times 's\ set\ )\ list \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$
where
$guards\ []\ c = c\ |$
$guards\ (g\#gs)\ c = Guard\ (fst\ g)\ (snd\ g)\ (guards\ gs\ c)$

definition

while:: $('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow ('s, 'p, 'f, 'e)\ com \Rightarrow ('s, 'p, 'f, 'e)\ com$
where
while gs b c = *guards gs* (*While b* (*Seq c* (*guards gs Skip*)))

definition
whileAnno::
$'s\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow ('s, 'p, 'f, 'e)\ com \Rightarrow ('s, 'p, 'f, 'e)\ com$
where
whileAnno b I V c = *While b c*

definition
whileAnnoG::
$('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow$
$('s, 'p, 'f, 'e)\ com \Rightarrow ('s, 'p, 'f, 'e)\ com$ **where**
whileAnnoG gs b I V c = *while gs b c*

definition
specAnno:: $('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s, 'p, 'f, 'e)\ com) \Rightarrow$
$('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('s, 'p, 'f, 'e)\ com$
where *specAnno P c Q A* = (*c undefined*)

definition
whileAnnoFix::
$'s\ bexp \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s \times 's)\ assn) \Rightarrow ('a \Rightarrow ('s, 'p, 'f, 'e)\ com)$
$\Rightarrow$
$('s, 'p, 'f, 'e)\ com$ **where**
whileAnnoFix b I V c = *While b* (*c undefined*)

definition
whileAnnoGFix::
$('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow ('a \Rightarrow 's\ assn) \Rightarrow ('a \Rightarrow ('s \times 's)\ assn) \Rightarrow$
$('a \Rightarrow ('s, 'p, 'f, 'e)\ com) \Rightarrow ('s, 'p, 'f, 'e)\ com$ **where**
whileAnnoGFix gs b I V c = *while gs b* (*c undefined*)

definition *if-rel*::$('s \Rightarrow bool) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \times 's)$
set
where *if-rel b f g h* = $\{(s,t).$ *if b s then* $t = f\ s$ *else* $t = g\ s \lor t = h\ s\}$

lemma *fst-guaranteeStripPair*: *fst* (*guaranteeStripPair f g*) = *f*
by (*simp add*: *guaranteeStripPair-def*)

lemma *snd-guaranteeStripPair*: *snd* (*guaranteeStripPair f g*) = *g*
by (*simp add*: *guaranteeStripPair-def*)

5.3 Operations on Simpl-Syntax

5.3.1 Normalisation of Sequential Composition: *sequence*, *flatten* and *normalize*

primrec *flatten*:: $('s, 'p, 'f, 'e)\ com \Rightarrow ('s, 'p, 'f, 'e)\ com\ list$

where
flatten Skip $=$ [*Skip*] |
flatten (*Basic f e*) $=$ [*Basic f e*] |
flatten (*Spec r e*) $=$ [*Spec r e*] |
flatten (*Seq* c_1 c_2) $=$ *flatten* c_1 @ *flatten* c_2 |
flatten (*Cond b* c_1 c_2) $=$ [*Cond b* c_1 c_2] |
flatten (*While b c*) $=$ [*While b c*] |
flatten (*Call p*) $=$ [*Call p*] |
flatten (*DynCom c*) $=$ [*DynCom c*] |
flatten (*Guard f g c*) $=$ [*Guard f g c*] |
flatten Throw $=$ [*Throw*] |
flatten (*Catch* c_1 c_2) $=$ [*Catch* c_1 c_2] |
flatten (*Await b ca e*) $=$ [*Await b ca e*]

primrec *flattenc*:: $('s,\ 'p,\ 'f,\ 'e)$ *com* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com list*
where
flattenc Skip $=$ [*Skip*] |
flattenc (*Basic f e*) $=$ [*Basic f e*] |
flattenc (*Spec r e*) $=$ [*Spec r e*] |
flattenc (*Seq* c_1 c_2) $=$ [*Seq* c_1 c_2] |
flattenc (*Cond b* c_1 c_2) $=$ [*Cond b* c_1 c_2] |
flattenc (*While b c*) $=$ [*While b c*] |
flattenc (*Call p*) $=$ [*Call p*] |
flattenc (*DynCom c*) $=$ [*DynCom c*] |
flattenc (*Guard f g c*) $=$ [*Guard f g c*] |
flattenc Throw $=$ [*Throw*] |
flattenc (*Catch* c_1 c_2) $=$ *flattenc* c_1 @ *flattenc* c_2 |
flattenc (*Await b ca e*) $=$ [*Await b ca e*]

primrec *sequence*:: $(('s,\ 'p,\ 'f,\ 'e)$ *com* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com*) $\Rightarrow$
$('s,\ 'p,\ 'f,\ 'e)$ *com list* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com*
where
sequence seq [] $=$ *Skip* |
sequence seq (*c#cs*) $=$ (*case cs of* [] $\Rightarrow$ *c*
| - $\Rightarrow$ *seq c* (*sequence seq cs*))

primrec *normalize*:: $('s,\ 'p,\ 'f,\ 'e)$ *com* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com*
where
normalize Skip $=$ *Skip* |
normalize (*Basic f e*) $=$ *Basic f e* |
normalize (*Spec r e*) $=$ *Spec r e* |
normalize (*Seq* c_1 c_2) $=$ *sequence Seq*
((*flatten* (*normalize* c_1)) @ (*flatten* (*normalize* c_2))) |
normalize (*Cond b* c_1 c_2) $=$ *Cond b* (*normalize* c_1) (*normalize* c_2) |
normalize (*While b c*) $=$ *While b* (*normalize c*) |
normalize (*Call p*) $=$ *Call p* |
normalize (*DynCom c*) $=$ *DynCom* ($\lambda s.$ (*normalize* (*c s*))) |

$normalize\ (Guard\ f\ g\ c) = Guard\ f\ g\ (normalize\ c)\ |$
$normalize\ Throw = Throw\ |$
$normalize\ (Catch\ c_1\ c_2) = Catch\ (normalize\ c_1)\ (normalize\ c_2)\ |$
$normalize\ (Await\ b\ ca\ e) = Await\ b\ (Language.normalize\ ca)\ e$

primrec $normalizec$:: $('s, 'p, 'f, 'e)\ com \Rightarrow ('s, 'p, 'f, 'e)\ com$
where
$normalizec\ Skip = Skip\ |$
$normalizec\ (Basic\ f\ e) = Basic\ f\ e\ |$
$normalizec\ (Spec\ r\ e) = Spec\ r\ e\ |$
$normalizec\ (Seq\ c_1\ c_2) = Seq\ (normalizec\ c_1)\ (normalizec\ c_2)\ |$
$normalizec\ (Cond\ b\ c_1\ c_2) = Cond\ b\ (normalizec\ c_1)\ (normalizec\ c_2)\ |$
$normalizec\ (While\ b\ c) = While\ b\ (normalizec\ c)\ |$
$normalizec\ (Call\ p) = Call\ p\ |$
$normalizec\ (DynCom\ c) = DynCom\ (\lambda s.\ (normalizec\ (c\ s)))\ |$
$normalizec\ (Guard\ f\ g\ c) = Guard\ f\ g\ (normalizec\ c)\ |$
$normalizec\ Throw = Throw\ |$
$normalizec\ (Catch\ c_1\ c_2) = sequence\ Catch$
$((flattenc\ (normalizec\ c_1))\ @\ (flattenc\ (normalizec\ c_2)))\ |$
$normalizec\ (Await\ b\ ca\ e) = Await\ b\ (Language.normalize\ ca)\ e$

lemma *flatten-nonEmpty*: $flatten\ c \neq []$
by (*induct c*) *simp-all*

lemma *flattenc-nonEmpty*: $flattenc\ c \neq []$
by (*induct c*) *simp-all*

lemma *flatten-single*: $\forall\, c \in set\ (flatten\ c').\ flatten\ c = [c]$
apply (*induct c'*)
apply *simp*
apply *simp*
apply *simp*
apply (*simp* (*no-asm-use*))
apply *blast*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply *simp*
done

lemma *flattenc-single*: $\forall\, c \in set\ (flattenc\ c').\ flattenc\ c = [c]$
apply (*induct c'*)
apply *simp*
apply *simp*
apply *simp*

apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply *blast*
apply *simp*
done

lemma *flatten-sequence-id*:
$[\![cs \neq []; \forall\, c \in set\ cs.\ flatten\ c = [c]]\!] \Longrightarrow flatten\ (sequence\ Seq\ cs) = cs$
apply (*induct cs*)
apply *simp*
apply (*case-tac cs*)
apply *simp*
apply *auto*
done

lemma *flattenc-sequence-id*:
$[\![cs \neq []; \forall\, c \in set\ cs.\ flattenc\ c = [c]]\!] \Longrightarrow flattenc\ (sequence\ Catch\ cs) = cs$
apply (*induct cs*)
apply *simp*
apply (*case-tac cs*)
apply *simp*
apply *auto*
done

lemma *flatten-app*:
$flatten\ (sequence\ Seq\ (flatten\ c1\ @\ flatten\ c2)) = flatten\ c1\ @\ flatten\ c2$
apply (*rule flatten-sequence-id*)
apply (*simp add*: *flatten-nonEmpty*)
apply (*simp*)
apply (*insert flatten-single*)
apply *blast*
done

lemma *flattenc-app*:
$flattenc\ (sequence\ Catch\ (flattenc\ c1\ @\ flattenc\ c2)) = flattenc\ c1\ @\ flattenc\ c2$
apply (*rule flattenc-sequence-id*)
apply (*simp add*: *flattenc-nonEmpty*)
apply (*simp*)
apply (*insert flattenc-single*)
apply *blast*
done

lemma *flatten-sequence-flatten*: *flatten* (*sequence Seq* (*flatten c*)) = *flatten c*
apply (*induct c*)
apply (*auto simp add*: *flatten-app*)
done

lemma *flattenc-sequence-flattenc*: *flattenc* (*sequence Catch* (*flattenc c*)) = *flattenc c*
apply (*induct c*)
apply (*auto simp add*: *flattenc-app*)
done

lemma *sequence-flatten-normalize*: *sequence Seq* (*flatten* (*normalize c*)) = *normalize c*
apply (*induct c*)
apply (*auto simp add*: *flatten-app*)
done

lemma *sequence-flattenc-normalize*: *sequence Catch* (*flattenc* (*normalizec c*)) = *normalizec c*
apply (*induct c*)
apply (*auto simp add*: *flattenc-app*)
done

lemma *flatten-normalize*: $\bigwedge$*x xs*. *flatten* (*normalize c*) = *x#xs*
$\Longrightarrow$ (*case xs of* [] $\Rightarrow$ *normalize c* = *x*
| (*x'#xs'*) $\Rightarrow$ *normalize c*= *Seq x* (*sequence Seq xs*))
proof (*induct c*)
case (*Seq c1 c2*)
have *flatten* (*normalize* (*Seq c1 c2*)) = *x # xs* **by** *fact*
hence *flatten* (*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*))) =
x#xs
by *simp*
hence *x-xs*: *flatten* (*normalize c1*) @ *flatten* (*normalize c2*) = *x # xs*
by (*simp add*: *flatten-app*)
show *?case*
proof (*cases flatten* (*normalize c1*))
case *Nil*
with *flatten-nonEmpty* **show** *?thesis* **by** *auto*
next
case (*Cons x1 xs1*)
note *Cons-x1-xs1* = *this*
with *x-xs* **obtain**
x-x1: *x*=*x1* **and** *xs-rest*: *xs*=*xs1*@*flatten* (*normalize c2*)
by *auto*

```
    show ?thesis
    proof (cases xs1)
      case Nil
      from Seq.hyps (1) [OF Cons-x1-xs1] Nil
      have normalize c1 = x1
        by simp
      with Cons-x1-xs1 Nil x-x1 xs-rest show ?thesis
        apply (cases flatten (normalize c2))
        apply (fastforce simp add: flatten-nonEmpty)
        apply simp
        done
    next
      case Cons
      from Seq.hyps (1) [OF Cons-x1-xs1] Cons
      have normalize c1 = Seq x1 (sequence Seq xs1)
        by simp
      with Cons-x1-xs1 Nil x-x1 xs-rest show ?thesis
        apply (cases flatten (normalize c2))
        apply (fastforce simp add: flatten-nonEmpty)
        apply (simp split: list.splits)
        done
    qed
  qed
qed (auto)

lemma flattenc-normalizec: ⋀x xs. flattenc (normalizec c) = x#xs
      ⟹ (case xs of [] ⇒ normalizec c = x
            | (x'#xs') ⇒ normalizec c= Catch x (sequence Catch xs))
proof (induct c)
  case (Catch c1 c2)
  have flattenc (normalizec (Catch c1 c2)) = x # xs by fact
  hence flattenc (sequence Catch (flattenc (normalizec c1) @ flattenc (normalizec
c2))) =
         x#xs
    by simp
  hence x-xs: flattenc (normalizec c1) @ flattenc (normalizec c2) = x # xs
    by (simp add: flattenc-app)
  show ?case
  proof (cases flattenc (normalizec c1))
    case Nil
    with flattenc-nonEmpty show ?thesis by auto
  next
    case (Cons x1 xs1)
    note Cons-x1-xs1 = this
    with x-xs obtain
      x-x1: x=x1 and xs-rest: xs=xs1@flattenc (normalizec c2)
      by auto
    show ?thesis
    proof (cases xs1)
```

    case *Nil*
    from *Catch.hyps* (*1*) [*OF Cons-x1-xs1*] *Nil*
    have *normalizec c1* = *x1*
      by *simp*
    with *Cons-x1-xs1 Nil x-x1 xs-rest* show *?thesis*
      apply (*cases flattenc* (*normalizec c2*))
      apply (*fastforce simp add*: *flattenc-nonEmpty*)
      apply *simp*
      done
  next
    case *Cons*
    from *Catch.hyps* (*1*) [*OF Cons-x1-xs1*] *Cons*
    have *normalizec c1* = *Catch x1* (*sequence Catch xs1*)
      by *simp*
    with *Cons-x1-xs1 Nil x-x1 xs-rest* show *?thesis*
      apply (*cases flattenc* (*normalizec c2*))
      apply (*fastforce simp add*: *flattenc-nonEmpty*)
      apply (*simp split*: *list.splits*)
      done
  qed
 qed
qed (*auto*)

lemma *flatten-raise* [*simp*]: *flatten* (*raise f e*) = [*Basic f e*, *Throw*]
 by (*simp add*: *raise-def*)

lemma *flatten-condCatch* [*simp*]: *flatten* (*condCatch c1 b c2*) = [*condCatch c1 b c2*]
 by (*simp add*: *condCatch-def*)

lemma *flatten-bind* [*simp*]: *flatten* (*bind e c*) = [*bind e c*]
 by (*simp add*: *bind-def*)

lemma *flatten-bseq* [*simp*]: *flatten* (*bseq c1 c2*) = *flatten c1* @ *flatten c2*
 by (*simp add*: *bseq-def*)

lemma *flatten-block* [*simp*]:
 flatten (*block init ei bdy return er result*) = [*block init ei bdy return er result*]
 by (*simp add*: *block-def*)

lemma *flatten-call* [*simp*]: *flatten* (*call init ei p return er result*) = [*call init ei p return er result*]
 by (*simp add*: *call-def*)

lemma *flatten-dynCall* [*simp*]: *flatten* (*dynCall init ei p return er result*) = [*dynCall init ei p return er result*]
 by (*simp add*: *dynCall-def*)

lemma *flatten-fcall* [*simp*]: *flatten* (*fcall init ei p return er result c*) = [*fcall init ei*

p return er result c]
by (*simp add*: *fcall-def*)

lemma *flatten-switch* [*simp*]: *flatten* (*switch v Vcs*) = [*switch v Vcs*]
by (*cases Vcs*) *auto*

lemma *flatten-guaranteeStrip* [*simp*]:
flatten (*guaranteeStrip f g c*) = [*guaranteeStrip f g c*]
by (*simp add*: *guaranteeStrip-def*)

lemma *flatten-while* [*simp*]: *flatten* (*while gs b c*) = [*while gs b c*]
apply (*simp add*: *while-def*)
apply (*induct gs*)
apply *auto*
done

lemma *flatten-whileAnno* [*simp*]:
flatten (*whileAnno b I V c*) = [*whileAnno b I V c*]
by (*simp add*: *whileAnno-def*)

lemma *flatten-whileAnnoG* [*simp*]:
flatten (*whileAnnoG gs b I V c*) = [*whileAnnoG gs b I V c*]
by (*simp add*: *whileAnnoG-def*)

lemma *flatten-specAnno* [*simp*]:
flatten (*specAnno P c Q A*) = *flatten* (*c undefined*)
by (*simp add*: *specAnno-def*)

lemmas *flatten-simps* = *flatten.simps flatten-raise flatten-condCatch flatten-bind*
flatten-block flatten-call flatten-dynCall flatten-fcall flatten-switch
flatten-guaranteeStrip
flatten-while flatten-whileAnno flatten-whileAnnoG flatten-specAnno

lemma *normalize-raise* [*simp*]:
normalize (*raise f e*) = *raise f e*
by (*simp add*: *raise-def*)

lemma *normalize-condCatch* [*simp*]:
normalize (*condCatch c1 b c2*) = *condCatch* (*normalize c1*) *b* (*normalize c2*)
by (*simp add*: *condCatch-def*)

lemma *normalize-bind* [*simp*]:
normalize (*bind e c*) = *bind e* (λ*v*. *normalize* (*c v*))
by (*simp add*: *bind-def*)

lemma *normalize-bseq* [*simp*]:
normalize (*bseq c1 c2*) = *sequence bseq*
((*flatten* (*normalize c1*)) @ (*flatten* (*normalize c2*)))
by (*simp add*: *bseq-def*)

lemma *normalize-block* [*simp*]: *normalize* (*block init ei bdy return er c*) =
block init ei (*normalize bdy*) *return er* (λ*s t*. *normalize* (*c s t*))
apply (*simp add*: *block-def*)
apply (*rule ext*)
apply (*simp*)
apply (*cases flatten* (*normalize bdy*))
apply (*simp add*: *flatten-nonEmpty*)
apply (*rule conjI*)
apply *simp*
apply (*drule flatten-normalize*)
apply (*case-tac list*)
apply *simp*
apply *simp*
apply (*rule ext*)
apply (*case-tac flatten* (*normalize* (*c s sa*)))
apply (*simp add*: *flatten-nonEmpty*)
apply *simp*
apply (*thin-tac flatten* (*normalize bdy*) = *P* **for** *P*)
apply (*drule flatten-normalize*)
apply (*case-tac lista*)
apply *simp*
apply *simp*
done

lemma *normalize-call* [*simp*]:
normalize (*call init ei p return er c*) = *call init ei p return er* (λ*i t*. *normalize* (*c i t*))
by (*simp add*: *call-def*)

lemma *normalize-dynCall* [*simp*]:
normalize (*dynCall init ei p return er c*) =
dynCall init ei p return er (λ*s t*. *normalize* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *normalize-fcall* [*simp*]:
normalize (*fcall init ei p return er result c*) =
fcall init ei p return er result (λ*v*. *normalize* (*c v*))
by (*simp add*: *fcall-def*)

lemma *normalize-switch* [*simp*]:
normalize (*switch v Vcs*) = *switch v* (*map* (λ(*V*,*c*). (*V*,*normalize c*)) *Vcs*)
apply (*induct Vcs*)
apply *auto*
done

lemma *normalize-guaranteeStrip* [*simp*]:
normalize (*guaranteeStrip f g c*) = *guaranteeStrip f g* (*normalize c*)
by (*simp add*: *guaranteeStrip-def*)

lemma *normalize-guards* [*simp*]:
normalize (*guards gs c*) = *guards gs* (*normalize c*)
by (*induct gs*) *auto*

Sequencial composition with guards in the body is not preserved by normalize

lemma *normalize-while* [*simp*]:
normalize (*while gs b c*) = *guards gs*
(*While b* (*sequence Seq* (*flatten* (*normalize c*) @ *flatten* (*guards gs Skip*))))
by (*simp add*: *while-def*)

lemma *normalize-whileAnno* [*simp*]:
normalize (*whileAnno b I V c*) = *whileAnno b I V* (*normalize c*)
by (*simp add*: *whileAnno-def*)

lemma *normalize-whileAnnoG* [*simp*]:
normalize (*whileAnnoG gs b I V c*) = *guards gs*
(*While b* (*sequence Seq* (*flatten* (*normalize c*) @ *flatten* (*guards gs Skip*))))
by (*simp add*: *whileAnnoG-def*)

lemma *normalize-specAnno* [*simp*]:
normalize (*specAnno P c Q A*) = *specAnno P* (λs. *normalize* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *normalize-simps* =
normalize.simps normalize-raise normalize-condCatch normalize-bind
normalize-block normalize-call normalize-dynCall normalize-fcall normalize-switch
normalize-guaranteeStrip normalize-guards
normalize-while normalize-whileAnno normalize-whileAnnoG normalize-specAnno

lemma *flattenc-raise* [*simp*]: *flattenc* (*raise f e*) = [*Seq* (*Basic f e*) *Throw*]
by (*simp add*: *raise-def*)

lemma *flattenc-condCatch* [*simp*]: *flattenc* (*condCatch c1 b c2*) = *flattenc c1* @ [*Cond b c2 Throw*]
by (*simp add*: *condCatch-def*)

lemma *flattenc-bind* [*simp*]: *flattenc* (*bind e c*) = [*bind e c*]
by (*simp add*: *bind-def*)

lemma *flattenc-bseq* [*simp*]: *flattenc* (*bseq c1 c2*) = [*Seq c1 c2*]
by (*simp add*: *bseq-def*)

lemma *flattenc-block* [*simp*]:
flattenc (*block init ei bdy return er result*) = [*block init ei bdy return er result*]
by (*simp add*: *block-def*)

lemma *flattenc-call* [*simp*]: *flattenc* (*call init ei p return er result*) = [*call init ei p return er result*]
  by (*simp add*: *call-def*)

lemma *flattenc-dynCall* [*simp*]: *flattenc* (*dynCall init ei p return er result*) = [*dynCall init ei p return er result*]
  by (*simp add*: *dynCall-def*)

lemma *flattenc-fcall* [*simp*]: *flattenc* (*fcall init ei p return er result c*) = [*fcall init ei p return er result c*]
  by (*simp add*: *fcall-def*)

lemma *flattenc-switch* [*simp*]: *flattenc* (*switch v Vcs*) = [*switch v Vcs*]
  by (*cases Vcs*) *auto*

lemma *flattenc-guaranteeStrip* [*simp*]:
  flattenc (*guaranteeStrip f g c*) = [*guaranteeStrip f g c*]
  by (*simp add*: *guaranteeStrip-def*)

lemma *flattenc-while* [*simp*]: *flattenc* (*while gs b c*) = [*while gs b c*]
  apply (*simp add*: *while-def*)
  apply (*induct gs*)
  apply *auto*
  done

lemma *flattenc-whileAnno* [*simp*]:
  flattenc (*whileAnno b I V c*) = [*whileAnno b I V c*]
  by (*simp add*: *whileAnno-def*)

lemma *flattenc-whileAnnoG* [*simp*]:
  flattenc (*whileAnnoG gs b I V c*) = [*whileAnnoG gs b I V c*]
  by (*simp add*: *whileAnnoG-def*)

lemma *flattenc-specAnno* [*simp*]:
  flattenc (*specAnno P c Q A*) = *flattenc* (*c undefined*)
  by (*simp add*: *specAnno-def*)

lemmas *flattenc-simps* = *flattenc.simps* *flattenc-condCatch* *flattenc-bind*
  flattenc-block *flattenc-call* *flattenc-dynCall* *flattenc-fcall* *flattenc-switch*
  flattenc-guaranteeStrip
  flattenc-while *flattenc-whileAnno* *flattenc-whileAnnoG* *flattenc-specAnno*

lemma *normalizec-raise*:
 normalizec (*raise f e*) = *raise f e*
  by (*simp add*: *raise-def*)

lemma *normalizec-condCatch*:
 normalizec (*condCatch c1 b c2*) = *sequence Catch* ((*flattenc* (*normalizec c1*))@ [*Cond b* (*normalizec c2*) *Throw*])

by (*simp add*: *condCatch-def*)

lemma *normalizec-bind*:
normalizec (*bind e c*) = *bind e* (λ*v*. *normalizec* (*c v*))
by (*simp add*: *bind-def*)

lemma *normalizec-bseq*:
normalizec (*bseq c1 c2*) = *bseq* (*normalizec c1*) (*normalizec c2*)
by (*simp add*: *bseq-def*)

lemma *normalizec-block*: *normalizec* (*block init ei bdy return er c*) =
block init ei (*normalizec bdy*) *return er* (λ*s t*. *normalizec* (*c s t*))
by (*simp add*: *block-def*)

lemma *normalizec-call*:
normalizec (*call init ei p return er c*) = *call init ei p return er* (λ*i t*. *normalizec* (*c i t*))
by (*simp add*: *call-def normalizec-block*)

lemma *normalizec-dynCall*:
normalizec (*dynCall init ei p return er c*) =
dynCall init ei p return er (λ*s t*. *normalizec* (*c s t*))
by (*simp add*: *dynCall-def normalizec-call*)

lemma *normalizec-fcall*:
normalizec (*fcall init ei p return er result c*) =
fcall init ei p return er result (λ*v*. *normalizec* (*c v*))
by (*simp add*: *fcall-def normalizec-call*)

lemma *normalizec-switch*:
normalizec (*switch v Vcs*) = *switch v* (*map* (λ(*V*,*c*). (*V*,*normalizec c*)) *Vcs*)
apply (*induct Vcs*)
apply *auto*
done

lemma *normalizec-guaranteeStrip*:
normalizec (*guaranteeStrip f g c*) = *guaranteeStrip f g* (*normalizec c*)
by (*simp add*: *guaranteeStrip-def*)

lemma *normalizec-guards*:
normalizec (*guards gs c*) = *guards gs* (*normalizec c*)
by (*induct gs*) *auto*

Sequencial composition with guards in the body is not preserved by normalize

lemma *normalizec-while*:
normalizec (*while gs b c*) = *guards gs*
(*While b* (*Seq* (*normalizec c*) (*guards gs Skip*)))

by (*simp add*: *while-def normalizec-guards*)

lemma *normalizec-whileAnno*:
normalizec (*whileAnno b I V c*) = *whileAnno b I V* (*normalizec c*)
by (*simp add*: *whileAnno-def*)

lemma *normalizec-whileAnnoG* :
normalizec (*whileAnnoG gs b I V c*) = *guards gs*
(*While b* (*Seq* (*normalizec c*) (*guards gs Skip*)))
by (*simp add*: *whileAnnoG-def normalizec-while*)

lemma *normalizec-specAnno*:
normalizec (*specAnno P c Q A*) = *specAnno P* (λs. *normalizec* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

5.3.2 Stripping Guards: *strip-guards*

primrec *strip-guards*:: *'f set* ⇒ (*'s*, *'p*, *'f*, *'e*) *com* ⇒ (*'s*, *'p*, *'f*, *'e*) *com*
where
strip-guards F Skip = *Skip* |
strip-guards F (*Basic f e*) = *Basic f e* |
strip-guards F (*Spec r e*) = *Spec r e* |
strip-guards F (*Seq* c_1 c_2) = (*Seq* (*strip-guards F* c_1) (*strip-guards F* c_2)) |
strip-guards F (*Cond b* c_1 c_2) = *Cond b* (*strip-guards F* c_1) (*strip-guards F* c_2) |
strip-guards F (*While b c*) = *While b* (*strip-guards F c*) |
strip-guards F (*Call p*) = *Call p* |
strip-guards F (*DynCom c*) = *DynCom* (λs. (*strip-guards F* (*c s*))) |
strip-guards F (*Guard f g c*) = (*if f* $\in$ *F then strip-guards F c*
else Guard f g (*strip-guards F c*)) |
strip-guards F Throw = *Throw* |
strip-guards F (*Catch* c_1 c_2) = *Catch* (*strip-guards F* c_1) (*strip-guards F* c_2) |
strip-guards F (*Await b ca e*) = *Await b* (*Language.strip-guards F ca*) *e*

lemma *no-await-strip-guards-eq*:
assumes *noawaits*:*noawaits t*
shows (*Language.strip-guards F* (*sequential t*)) = (*sequential* (*strip-guards F t*))
using *noawaits*
by (*induct t*) *auto*

definition *strip*:: *'f set* ⇒
(*'p* ⇒ (*'s*, *'p*, *'f*, *'e*) *com option*) ⇒ (*'p* ⇒ (*'s*, *'p*, *'f*, *'e*) *com option*)
where *strip F* Γ = (λp. *map-option* (*strip-guards F*) (Γ *p*))

lemma *strip-simp* [*simp*]: (*strip F* Γ) *p* = *map-option* (*strip-guards F*) (Γ *p*)
by (*simp add*: *strip-def*)

lemma *dom-strip*: *dom* (*strip F* Γ) = *dom* Γ
by (*auto*)

lemma *strip-guards-idem*: *strip-guards F* (*strip-guards F c*) = *strip-guards F c*
by (*induct c*) (*auto simp add*:*Language.strip-guards-idem*)

lemma *strip-idem*: *strip F* (*strip F* Γ) = *strip F* Γ
apply (*rule ext*)
apply (*case-tac* Γ *x*)
apply (*auto simp add*: *strip-guards-idem strip-def*)
done

lemma *strip-guards-raise* [*simp*]:
strip-guards F (*raise f e*) = *raise f e*
by (*simp add*: *raise-def*)

lemma *strip-guards-condCatch* [*simp*]:
strip-guards F (*condCatch c1 b c2*) =
condCatch (*strip-guards F c1*) *b* (*strip-guards F c2*)
by (*simp add*: *condCatch-def*)

lemma *strip-guards-bind* [*simp*]:
strip-guards F (*bind e c*) = *bind e* (λv. *strip-guards F* (*c v*))
by (*simp add*: *bind-def*)

lemma *strip-guards-bseq* [*simp*]:
strip-guards F (*bseq c1 c2*) = *bseq* (*strip-guards F c1*) (*strip-guards F c2*)
by (*simp add*: *bseq-def*)

lemma *strip-guards-block* [*simp*]:
strip-guards F (*block init ei bdy return er c*) =
block init ei (*strip-guards F bdy*) *return er* (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *block-def*)

lemma *strip-guards-call* [*simp*]:
strip-guards F (*call init ei p return er c*) =
call init ei p return er (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *call-def*)

lemma *strip-guards-dynCall* [*simp*]:
strip-guards F (*dynCall init ei p return er c*) =
dynCall init ei p return er (λs *t*. *strip-guards F* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *strip-guards-fcall* [*simp*]:
strip-guards F (*fcall init ei p return er result c*) =

fcall init ei p return er result (λv. *strip-guards F* (*c v*))
by (*simp add*: *fcall-def*)

lemma *strip-guards-switch* [*simp*]:
strip-guards F (*switch v Vc*) =
switch v (*map* (λ(*V*,*c*). (*V*,*strip-guards F c*)) *Vc*)
by (*induct Vc*) *auto*

lemma *strip-guards-guaranteeStrip* [*simp*]:
strip-guards F (*guaranteeStrip f g c*) =
(*if f* $\in$ *F then strip-guards F c*
else guaranteeStrip f g (*strip-guards F c*))
by (*simp add*: *guaranteeStrip-def*)

lemma *guaranteeStripPair-split-conv* [*simp*]: *case-prod c* (*guaranteeStripPair f g*) = *c f g*
by (*simp add*: *guaranteeStripPair-def*)

lemma *strip-guards-guards* [*simp*]: *strip-guards F* (*guards gs c*) =
guards (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) (*strip-guards F c*)
by (*induct gs*) *auto*

lemma *strip-guards-while* [*simp*]:
strip-guards F (*while gs b c*) =
while (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) *b* (*strip-guards F c*)
by (*simp add*: *while-def*)

lemma *strip-guards-whileAnno* [*simp*]:
strip-guards F (*whileAnno b I V c*) = *whileAnno b I V* (*strip-guards F c*)
by (*simp add*: *whileAnno-def while-def*)

lemma *strip-guards-whileAnnoG* [*simp*]:
strip-guards F (*whileAnnoG gs b I V c*) =
whileAnnoG (*filter* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) *b I V* (*strip-guards F c*)
by (*simp add*: *whileAnnoG-def*)

lemma *strip-guards-specAnno* [*simp*]:
strip-guards F (*specAnno P c Q A*) =
specAnno P (λs. *strip-guards F* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *strip-guards-simps* = *strip-guards.simps strip-guards-raise*
strip-guards-condCatch strip-guards-bind strip-guards-bseq strip-guards-block
strip-guards-dynCall strip-guards-fcall strip-guards-switch
strip-guards-guaranteeStrip guaranteeStripPair-split-conv strip-guards-guards
strip-guards-while strip-guards-whileAnno strip-guards-whileAnnoG
strip-guards-specAnno

5.3.3 Marking Guards: *mark-guards*

primrec *mark-guards*:: $'f \Rightarrow ('s,'p,'g,\ 'e)\ com \Rightarrow ('s,\ 'p,\ 'f,\ 'e)\ com$
where
mark-guards f Skip = *Skip* |
mark-guards f (*Basic g e*) = *Basic g e* |
mark-guards f (*Spec r e*) = *Spec r e* |
mark-guards f (*Seq* c_1 c_2) = (*Seq* (*mark-guards f* c_1) (*mark-guards f* c_2)) |
mark-guards f (*Cond b* c_1 c_2) = *Cond b* (*mark-guards f* c_1) (*mark-guards f* c_2) |
mark-guards f (*While b c*) = *While b* (*mark-guards f c*) |
mark-guards f (*Call p*) = *Call p* |
mark-guards f (*DynCom c*) = *DynCom* ($\lambda s.$ (*mark-guards f* (*c s*))) |
mark-guards f (*Guard f' g c*) = *Guard f g* (*mark-guards f c*) |
mark-guards f Throw = *Throw* |
mark-guards f (*Catch* c_1 c_2) = *Catch* (*mark-guards f* c_1) (*mark-guards f* c_2) |
mark-guards f (*Await b ca e*) = *Await b* (*Language.mark-guards f ca*) *e*

lemma *mark-guards-raise*: *mark-guards f* (*raise g e*) = *raise g e*
by (*simp add*: *raise-def*)

lemma *mark-guards-condCatch* [*simp*]:
mark-guards f (*condCatch c1 b c2*) =
condCatch (*mark-guards f c1*) *b* (*mark-guards f c2*)
by (*simp add*: *condCatch-def*)

lemma *mark-guards-bind* [*simp*]:
mark-guards f (*bind e c*) = *bind e* ($\lambda v.$ *mark-guards f* (*c v*))
by (*simp add*: *bind-def*)

lemma *mark-guards-bseq* [*simp*]:
mark-guards f (*bseq c1 c2*) = *bseq* (*mark-guards f c1*) (*mark-guards f c2*)
by (*simp add*: *bseq-def*)

lemma *mark-guards-block* [*simp*]:
mark-guards f (*block init ei bdy return er c*) =
block init ei (*mark-guards f bdy*) *return er* ($\lambda s\ t.$ *mark-guards f* (*c s t*))
by (*simp add*: *block-def*)

lemma *mark-guards-call* [*simp*]:
mark-guards f (*call init ei p return er c*) =
call init ei p return er ($\lambda s\ t.$ *mark-guards f* (*c s t*))
by (*simp add*: *call-def*)

lemma *mark-guards-dynCall* [*simp*]:
mark-guards f (*dynCall init ei p return er c*) =
dynCall init ei p return er ($\lambda s\ t.$ *mark-guards f* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *mark-guards-fcall* [*simp*]:
mark-guards f (*fcall init ei p return er result c*) =

fcall init ei p return er result (λv. *mark-guards f* (*c v*))
by (*simp add*: *fcall-def*)

lemma *mark-guards-switch* [*simp*]:
mark-guards f (*switch v vs*) =
switch v (*map* (λ(*V*,*c*). (*V*,*mark-guards f c*)) *vs*)
by (*induct vs*) *auto*

lemma *mark-guards-guaranteeStrip* [*simp*]:
mark-guards f (*guaranteeStrip f' g c*) = *guaranteeStrip f g* (*mark-guards f c*)
by (*simp add*: *guaranteeStrip-def*)

lemma *mark-guards-guards* [*simp*]:
mark-guards f (*guards gs c*) = *guards* (*map* (λ(*f'*,*g*). (*f*,*g*)) *gs*) (*mark-guards f c*)
by (*induct gs*) *auto*

lemma *mark-guards-while* [*simp*]:
mark-guards f (*while gs b c*) =
while (*map* (λ(*f'*,*g*). (*f*,*g*)) *gs*) *b* (*mark-guards f c*)
by (*simp add*: *while-def*)

lemma *mark-guards-whileAnno* [*simp*]:
mark-guards f (*whileAnno b I V c*) = *whileAnno b I V* (*mark-guards f c*)
by (*simp add*: *whileAnno-def while-def*)

lemma *mark-guards-whileAnnoG* [*simp*]:
mark-guards f (*whileAnnoG gs b I V c*) =
whileAnnoG (*map* (λ(*f'*,*g*). (*f*,*g*)) *gs*) *b I V* (*mark-guards f c*)
by (*simp add*: *whileAnno-def whileAnnoG-def while-def*)

lemma *mark-guards-specAnno* [*simp*]:
mark-guards f (*specAnno P c Q A*) =
specAnno P (λs. *mark-guards f* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

lemmas *mark-guards-simps* = *mark-guards.simps mark-guards-raise*
mark-guards-condCatch mark-guards-bind mark-guards-bseq mark-guards-block
mark-guards-dynCall mark-guards-fcall mark-guards-switch
mark-guards-guaranteeStrip guaranteeStripPair-split-conv mark-guards-guards
mark-guards-while mark-guards-whileAnno mark-guards-whileAnnoG
mark-guards-specAnno

definition *is-Guard*:: ('*s*, '*p*, '*f*, '*e*) *com* ⇒ *bool*
where *is-Guard c* = (*case c of Guard f g c'* ⇒ *True* | - ⇒ *False*)
lemma *is-Guard-basic-simps* [*simp*]:
is-Guard Skip = *False*
is-Guard (*Basic f ev*) = *False*
is-Guard (*Spec r ev*) = *False*

is-Guard (*Seq c1 c2*) = *False*
is-Guard (*Cond b c1 c2*) = *False*
is-Guard (*While b c*) = *False*
is-Guard (*Call p*) = *False*
is-Guard (*DynCom C*) = *False*
is-Guard (*Guard F g c*) = *True*
is-Guard (*Throw*) = *False*
is-Guard (*Catch c1 c2*) = *False*
is-Guard (*raise f ev*) = *False*
is-Guard (*condCatch c1 b c2*) = *False*
is-Guard (*bind e cv*) = *False*
is-Guard (*bseq c1 c2*) = *False*
is-Guard (*block init ei bdy return er cont*) = *False*
is-Guard (*call init ei p return er cont*) = *False*
is-Guard (*dynCall init ei P return er cont*) = *False*
is-Guard (*fcall init ei p return er result cont'*) = *False*
is-Guard (*whileAnno b I V c*) = *False*
is-Guard (*guaranteeStrip F g c*) = *True*
is-Guard (*Await b ca ev*) = *False*
by (*auto simp add*: *is-Guard-def raise-def condCatch-def bind-def bseq-def block-def call-def dynCall-def fcall-def whileAnno-def guaranteeStrip-def*)

lemma *is-Guard-switch* [*simp*]:
is-Guard (*switch v Vc*) = *False*
by (*induct Vc*) *auto*

lemmas *is-Guard-simps* = *is-Guard-basic-simps is-Guard-switch*

primrec *dest-Guard*:: (*'s*, *'p*, *'f*, *'e*) *com* ⇒ (*'f* × *'s set* × (*'s*, *'p*, *'f*, *'e*) *com*)
where *dest-Guard* (*Guard f g c*) = (*f*,*g*,*c*)

lemma *dest-Guard-guaranteeStrip* [*simp*]: *dest-Guard* (*guaranteeStrip f g c*) = (*f*,*g*,*c*)
by (*simp add*: *guaranteeStrip-def*)

lemmas *dest-Guard-simps* = *dest-Guard.simps dest-Guard-guaranteeStrip*

5.3.4 Merging Guards: *merge-guards*

primrec *merge-guards*:: (*'s*, *'p*, *'f*, *'e*) *com* ⇒ (*'s*, *'p*, *'f*, *'e*) *com*
where
merge-guards Skip = *Skip* |
merge-guards (*Basic g e*) = *Basic g e* |
merge-guards (*Spec r e*) = *Spec r e* |
merge-guards (*Seq* c_1 c_2) = (*Seq* (*merge-guards* c_1) (*merge-guards* c_2)) |
merge-guards (*Cond b* c_1 c_2) = *Cond b* (*merge-guards* c_1) (*merge-guards* c_2) |
merge-guards (*While b c*) = *While b* (*merge-guards c*) |
merge-guards (*Call p*) = *Call p* |

merge-guards (*DynCom* *c*) = *DynCom* (λ*s*. (*merge-guards* (*c* *s*))) |
merge-guards (*Await* *b* *ca* *e*) = *Await* *b* (*Language.merge-guards* *ca*) *e* |

merge-guards (*Guard* *f* *g* *c*) =
  (*let* *c'* = (*merge-guards* *c*)
   in *if* *is-Guard* *c'*
    then *let* (*f'*,*g'*,*c''*) = *dest-Guard* *c'*
      in *if* *f*=*f'* *then* *Guard* *f* (*g* $\cap$ *g'*) *c''*
        else *Guard* *f* *g* (*Guard* *f'* *g'* *c''*)
    else *Guard* *f* *g* *c'*) |
merge-guards *Throw* = *Throw* |
merge-guards (*Catch* c_1 c_2) = *Catch* (*merge-guards* c_1) (*merge-guards* c_2)

lemma *merge-guards-res-Skip*: *merge-guards* *c* = *Skip* $\Longrightarrow$ *c* = *Skip*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Basic*: *merge-guards* *c* = *Basic* *f* *e* $\Longrightarrow$ *c* = *Basic* *f* *e*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Spec*: *merge-guards* *c* = *Spec* *r* *e* $\Longrightarrow$ *c* = *Spec* *r* *e*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Seq*: *merge-guards* *c* = *Seq* *c1* *c2* $\Longrightarrow$
  $\exists$ *c1'* *c2'*. *c* = *Seq* *c1'* *c2'* $\wedge$ *merge-guards* *c1'* = *c1* $\wedge$ *merge-guards* *c2'* = *c2*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Cond*: *merge-guards* *c* = *Cond* *b* *c1* *c2* $\Longrightarrow$
  $\exists$ *c1'* *c2'*. *c* = *Cond* *b* *c1'* *c2'* $\wedge$ *merge-guards* *c1'* = *c1* $\wedge$ *merge-guards* *c2'* =
c2
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-While*: *merge-guards* *c* = *While* *b* *c'* $\Longrightarrow$
  $\exists$ *c''*. *c* = *While* *b* *c''* $\wedge$ *merge-guards* *c''* = *c'*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Call*: *merge-guards* *c* = *Call* *p* $\Longrightarrow$ *c* = *Call* *p*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-DynCom*: *merge-guards* *c* = *DynCom* *c'* $\Longrightarrow$
  $\exists$ *c''*. *c* = *DynCom* *c''* $\wedge$ (λ*s*. (*merge-guards* (*c''* *s*))) = *c'*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Throw*: *merge-guards* *c* = *Throw* $\Longrightarrow$ *c* = *Throw*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Catch*: *merge-guards* *c* = *Catch* *c1* *c2* $\Longrightarrow$
  $\exists$ *c1'* *c2'*. *c* = *Catch* *c1'* *c2'* $\wedge$ *merge-guards* *c1'* = *c1* $\wedge$ *merge-guards* *c2'* = *c2*
 by (*cases* *c*) (*auto* *split*: *com.splits* *if-split-asm* *simp* *add*: *is-Guard-def* *Let-def*)

lemma *merge-guards-res-Guard*:
merge-guards c = *Guard f g c'* $\Longrightarrow$ $\exists$ *c'' f' g'*. *c* = *Guard f' g' c''*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Await*: *merge-guards c* = *Await b c' e* $\Longrightarrow$
$\exists$ *c''*. *c* = *Await b c'' e* $\wedge$ *Language.merge-guards c''* = *c'*
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemmas *merge-guards-res-simps* = *merge-guards-res-Skip merge-guards-res-Basic*
merge-guards-res-Spec merge-guards-res-Seq merge-guards-res-Cond
merge-guards-res-While merge-guards-res-Call
merge-guards-res-DynCom merge-guards-res-Throw merge-guards-res-Catch
merge-guards-res-Guard merge-guards-res-Await

lemma *merge-guards-raise*: *merge-guards* (*raise g e*) = *raise g e*
by (*simp add*: *raise-def*)

lemma *merge-guards-condCatch* [*simp*]:
merge-guards (*condCatch c1 b c2*) =
condCatch (*merge-guards c1*) *b* (*merge-guards c2*)
by (*simp add*: *condCatch-def*)

lemma *merge-guards-bind* [*simp*]:
merge-guards (*bind e c*) = *bind e* (λv. *merge-guards* (*c v*))
by (*simp add*: *bind-def*)

lemma *merge-guards-bseq* [*simp*]:
merge-guards (*bseq c1 c2*) = *bseq* (*merge-guards c1*) (*merge-guards c2*)
by (*simp add*: *bseq-def*)

lemma *merge-guards-block* [*simp*]:
merge-guards (*block init ei bdy return er c*) =
block init ei (*merge-guards bdy*) *return er* (λs *t*. *merge-guards* (*c s t*))
by (*simp add*: *block-def*)

lemma *merge-guards-call* [*simp*]:
merge-guards (*call init ei p return er c*) =
call init ei p return er (λs *t*. *merge-guards* (*c s t*))
by (*simp add*: *call-def*)

lemma *merge-guards-dynCall* [*simp*]:
merge-guards (*dynCall init ei p return er c*) =
dynCall init ei p return er (λs *t*. *merge-guards* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *merge-guards-fcall* [*simp*]:
merge-guards (*fcall init ei p return er result c*) =

fcall init ei p return er result ($\lambda v.$ *merge-guards* (*c v*))
by (*simp add*: *fcall-def*)

lemma *merge-guards-switch* [*simp*]:
merge-guards (*switch v vs*) =
switch v (*map* (λ(*V*,*c*). (*V*,*merge-guards c*)) *vs*)
by (*induct vs*) *auto*

lemma *merge-guards-guaranteeStrip* [*simp*]:
merge-guards (*guaranteeStrip f g c*) =
(*let* c' = (*merge-guards c*)
in if is-Guard c'
then let (f',g',c') = *dest-Guard* c'
in if f=f' *then Guard f* ($g \cap g'$) c'
else Guard f g (*Guard* f' g' c')
else Guard f g c')
by (*simp add*: *guaranteeStrip-def*)

lemma *merge-guards-whileAnno* [*simp*]:
merge-guards (*whileAnno b I V c*) = *whileAnno b I V* (*merge-guards c*)
by (*simp add*: *whileAnno-def while-def*)

lemma *merge-guards-specAnno* [*simp*]:
merge-guards (*specAnno P c Q A*) =
specAnno P ($\lambda s.$ *merge-guards* (*c undefined*)) *Q A*
by (*simp add*: *specAnno-def*)

LanguageCon.merge-guards for guard-lists as in *LanguageCon.guards*, *LanguageCon.while* and *LanguageCon.whileAnnoG* may have funny effects since the guard-list has to be merged with the body statement too.

lemmas *merge-guards-simps* = *merge-guards.simps merge-guards-raise*
merge-guards-condCatch merge-guards-bind merge-guards-bseq merge-guards-block
merge-guards-dynCall merge-guards-fcall merge-guards-switch
merge-guards-guaranteeStrip merge-guards-whileAnno merge-guards-specAnno

primrec *noguards*:: ($'s$, $'p$, $'f$, $'e$) *com* $\Rightarrow$ *bool*
where
noguards Skip = *True* |
noguards (*Basic f e*) = *True* |
noguards (*Spec r e*) = *True* |
noguards (*Seq* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2) |
noguards (*Cond b* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2) |
noguards (*While b c*) = (*noguards c*) |
noguards (*Call p*) = *True* |
noguards (*DynCom c*) = ($\forall s.$ *noguards* (*c s*)) |
noguards (*Guard f g c*) = *False* |
noguards Throw = *True* |
noguards (*Catch* c_1 c_2) = (*noguards* c_1 $\wedge$ *noguards* c_2) |
noguards (*Await b c e*) = (*Language.noguards c*)

lemma *noawaits-noguards-seq*:*noawaits* $c \Longrightarrow$ *noguards* $c =$ *Language.noguards* (*sequential* c)
by (*induct* c, *auto*)

lemma *noguards-strip-guards*: *noguards* (*strip-guards UNIV* c)
by (*induct* c) (*auto simp add*: *noguards-strip-guards*)

primrec *nothrows*:: $('s, 'p, 'f, 'e)$ *com* $\Rightarrow$ *bool*
where
nothrows Skip $=$ *True* |
nothrows (*Basic f e*) $=$ *True* |
nothrows (*Spec r e*) $=$ *True* |
nothrows (*Seq* c_1 c_2) $=$ (*nothrows* c_1 $\wedge$ *nothrows* c_2) |
nothrows (*Cond b* c_1 c_2) $=$ (*nothrows* c_1 $\wedge$ *nothrows* c_2) |
nothrows (*While b c*) $=$ *nothrows c* |
nothrows (*Call p*) $=$ *True* |
nothrows (*DynCom c*) $=$ ($\forall s$. *nothrows* (*c s*)) |
nothrows (*Guard f g c*) $=$ *nothrows c* |
nothrows Throw $=$ *False* |
nothrows (*Catch* c_1 c_2) $=$ (*nothrows* c_1 $\wedge$ *nothrows* c_2) |
nothrows (*Await b cn e*) $=$ *Language.nothrows cn*

lemma *noawaits-nothrows-seq*:*noawaits* $c \Longrightarrow$ *nothrows* $c =$ *Language.nothrows* (*sequential* c)
by (*induct* c, *auto*)

5.3.5 Intersecting Guards: $c_1 \cap_g c_2$

inductive-set *com-rel* ::$(('s, 'p, 'f, 'e)$ *com* $\times$ $('s, 'p, 'f, 'e)$ *com*) *set*
where
(*c1*, *Seq c1 c2*) $\in$ *com-rel*
| (*c2*, *Seq c1 c2*) $\in$ *com-rel*
| (*c1*, *Cond b c1 c2*) $\in$ *com-rel*
| (*c2*, *Cond b c1 c2*) $\in$ *com-rel*
| (*c*, *While b c*) $\in$ *com-rel*
| (*c x*, *DynCom c*) $\in$ *com-rel*
| (*c*, *Guard f g c*) $\in$ *com-rel*
| (*c1*, *Catch c1 c2*) $\in$ *com-rel*
| (*c2*, *Catch c1 c2*) $\in$ *com-rel*

inductive-cases *com-rel-elim-cases*:
(*c*, *Skip*) $\in$ *com-rel*
(*c*, *Basic f e*) $\in$ *com-rel*
(*c*, *Spec r e*) $\in$ *com-rel*
(*c*, *Seq c1 c2*) $\in$ *com-rel*
(*c*, *Cond b c1 c2*) $\in$ *com-rel*

$(c,$ *While b c1*$) \in$ *com-rel*
$(c,$ *Call p*$) \in$ *com-rel*
$(c,$ *DynCom c1*$) \in$ *com-rel*
$(c,$ *Guard f g c1*$) \in$ *com-rel*
$(c,$ *Throw*$) \in$ *com-rel*
$(c,$ *Catch c1 c2*$) \in$ *com-rel*
$(c,$ *Await b cn e*$) \in$ *com-rel*

lemma *wf-com-rel*: *wf com-rel*
apply (*rule wfUNIVI*)
apply (*induct-tac x*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
done

consts *inter-guards*:: $('s,\ 'p,\ 'f,\ 'e)$ *com* $\times$ $('s,\ 'p,\ 'f,\ 'e)$ *com* $\Rightarrow$ $('s,\ 'p,\ 'f,\ 'e)$ *com option*

abbreviation
inter-guards-syntax :: $('s,'p,'f,'e)$ *LanguageCon.com* $\Rightarrow$ $('s,'p,'f,'e)$ *LanguageCon.com* $\Rightarrow$ $('s,'p,'f,'e)$ *LanguageCon.com option*
(- $\cap_{gs}$ - [*20*,*20*] *19*)
where ((*c*::$('s,\ 'p,\ 'f,\ 'e)$ *com*) $\cap_{gs}$ (*d*::$('s,\ 'p,\ 'f,\ 'e)$ *com*)) == *LanguageCon.inter-guards* (*c*,*d*)

recdef *inter-guards inv-image com-rel fst*
(*Skip* $\cap_{gs}$ *Skip*) = *Some Skip*

(*Basic f1 e1* $\cap_{gs}$ *Basic f2 e2*) = (*if* (*f1*=*f2*) $\wedge$ (*e1*=*e2*) *then Some* (*Basic f1 e1*) *else None*)
(*Spec r1 e1* $\cap_{gs}$ *Spec r2 e2*) = (*if* (*r1*=*r2*) $\wedge$ (*e1*=*e2*) *then Some* (*Spec r1 e1*) *else None*)
(*Seq a1 a2* $\cap_{gs}$ *Seq b1 b2*) =
(*case* (*a1* $\cap_{gs}$ *b1*) *of*
None $\Rightarrow$ *None*
| *Some c1* $\Rightarrow$ (*case* (*a2* $\cap_{gs}$ *b2*) *of*

```
                  None ⇒ None
                | Some c2 ⇒ Some (Seq c1 c2)))

(Cond cnd1 t1 e1 ∩_gs Cond cnd2 t2 e2) =
   (if (cnd1=cnd2)
    then (case (t1 ∩_gs t2) of
            None ⇒ None
          | Some t ⇒ (case (e1 ∩_gs e2) of
                         None ⇒ None
                       | Some e ⇒ Some (Cond cnd1 t e)))
    else None)

(While cnd1 c1 ∩_gs While cnd2 c2) =
    (if (cnd1=cnd2 )
     then (case (c1 ∩_gs c2) of
             None ⇒ None
           | Some c ⇒ Some (While cnd1 c))
     else None)

(Call p1 ∩_gs Call p2) =
   (if p1 = p2
    then Some (Call p1)
    else None)

(DynCom P1 ∩_gs DynCom P2) =
   (if (∀ s. ((P1 s) ∩_gs (P2 s)) ≠ None)
   then Some (DynCom (λs. the ((P1 s) ∩_gs (P2 s))))
   else None)

(Guard m1 g1 c1 ∩_gs Guard m2 g2 c2) =
   (if m1=m2 then
       (case (c1 ∩_gs c2) of
          None ⇒ None
        | Some c ⇒ Some (Guard m1 (g1 ∩ g2) c))
    else None)

(Throw ∩_gs Throw) = Some Throw
(Catch a1 a2 ∩_gs Catch b1 b2) =
   (case (a1 ∩_gs b1) of
      None ⇒ None
    | Some c1 ⇒ (case (a2 ∩_gs b2) of
                  None ⇒ None
                | Some c2 ⇒ Some (Catch c1 c2)))
(Await cnd1 ca1 e1 ∩_gs Await cnd2 ca2 e2) =
 (if (cnd1=cnd2 ∧ e1=e2) then
       (case (ca1 ∩_g ca2) of
             None ⇒ None
           | Some c ⇒ Some (Await cnd1 c e1))
     else None)
```

$(c \cap_{gs} d) = None$

(**hints** *cong add*: *option.case-cong if-cong*
recdef-wf: *wf-com-rel simp*: *com-rel.intros*)

lemma *inter-guards-strip-eq*:
$\bigwedge(c::('s, 'p, 'f, 'e)\ com).\ ((c1::('s, 'p, 'f, 'e)\ com) \cap_{gs} (c2::('s, 'p, 'f, 'e)\ com))$
$= Some\ c \implies$
$(strip\text{-}guards\ UNIV\ c = strip\text{-}guards\ UNIV\ c1) \wedge$
$(strip\text{-}guards\ UNIV\ c = strip\text{-}guards\ UNIV\ c2)$
apply (*induct c1 c2 rule*: *inter-guards.induct*)
prefer *8*
apply (*simp split*: *if-split-asm*)
apply *hypsubst*
apply *simp*
apply (*rule ext*)
apply (*erule-tac x=s* **in** *allE*, *erule exE*)
apply (*erule-tac x=s* **in** *allE*)
apply *fastforce*
apply (*fastforce dest*:*inter-guards-strip-eq split*: *option.splits if-split-asm*)+
done

lemma *inter-guards-sym*: $\bigwedge c.\ (c1 \cap_{gs} c2) = Some\ c \Longrightarrow (c2 \cap_{gs} c1) = Some\ c$
apply (*induct c1 c2 rule*: *inter-guards.induct*)
apply (*simp-all*)
prefer *7*
apply (*simp split*: *if-split-asm*)
apply (*rule conjI*)
apply (*clarsimp*)
apply (*rule ext*)
apply (*erule-tac x=s* **in** *allE*)+
apply (*fastforce dest*:*inter-guards-sym split*: *option.splits if-split-asm*)+
done

lemma *inter-guards-Skip*: $(Skip \cap_{gs} c2) = Some\ c = (c2{=}Skip \wedge c{=}Skip)$
by (*cases c2*) *auto*

lemma *inter-guards-Basic*:
$((Basic\ f\ e1) \cap_{gs} c2) = Some\ c = (c2{=}Basic\ f\ e1 \wedge c{=}Basic\ f\ e1)$
by (*cases c2*) *auto*

lemma *inter-guards-Spec*:
$((Spec\ r\ e1) \cap_{gs} c2) = Some\ c = (c2{=}Spec\ r\ e1 \wedge c{=}Spec\ r\ e1)$
by (*cases c2*) *auto*

lemma *inter-guards-Seq*:
$(Seq\ a1\ a2 \cap_{gs} c2) = Some\ c =$

$(\exists\, b1\ b2\ d1\ d2.\ c2{=}Seq\ b1\ b2 \wedge (a1 \cap_{gs} b1) = Some\ d1 \wedge$
$(a2 \cap_{gs} b2) = Some\ d2 \wedge c{=}Seq\ d1\ d2)$
by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-Cond*:
$(Cond\ cnd\ t1\ e1 \cap_{gs} c2) = Some\ c =$
$(\exists\, t2\ e2\ t\ e.\ c2{=}Cond\ cnd\ t2\ e2 \wedge (t1 \cap_{gs} t2) = Some\ t \wedge$
$(e1 \cap_{gs} e2) = Some\ e \wedge c{=}Cond\ cnd\ t\ e)$
by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-While*:
$(While\ cnd\ bdy1 \cap_{gs} c2) = Some\ c =$
$(\exists\, bdy2\ bdy.\ c2 = While\ cnd\ bdy2 \wedge (bdy1 \cap_{gs} bdy2) = Some\ bdy \wedge$
$c{=}While\ cnd\ bdy)$
by (*cases c2*) (*auto split*: *option.splits if-split-asm*)

lemma *inter-guards-Await*:
$(Await\ cnd\ bdy1\ e1 \cap_{gs} c2) = Some\ c =$
$(\exists\, bdy2\ bdy.\ c2 = Await\ cnd\ bdy2\ e1 \wedge (bdy1 \cap_{g} bdy2) = Some\ bdy \wedge$
$c{=}Await\ cnd\ bdy\ e1)$
by (*cases c2*) (*auto split*: *option.splits if-split-asm*)

lemma *inter-guards-Call*:
$(Call\ p \cap_{gs} c2) = Some\ c =$
$(c2{=}Call\ p \wedge c{=}Call\ p)$
by (*cases c2*) (*auto split*: *if-split-asm*)

lemma *inter-guards-DynCom*:
$(DynCom\ f1 \cap_{gs} c2) = Some\ c =$
$(\exists\, f2.\ c2{=}DynCom\ f2 \wedge (\forall\, s.\ ((f1\ s) \cap_{gs} (f2\ s)) \neq None) \wedge$
$c{=}DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_{gs} (f2\ s))))$
by (*cases c2*) (*auto split*: *if-split-asm*)

lemma *inter-guards-Guard*:
$(Guard\ f\ g1\ bdy1 \cap_{gs} c2) = Some\ c =$
$(\exists\, g2\ bdy2\ bdy.\ c2{=}Guard\ f\ g2\ bdy2 \wedge (bdy1 \cap_{gs} bdy2) = Some\ bdy \wedge$
$c{=}Guard\ f\ (g1 \cap g2)\ bdy)$
by (*cases c2*) (*auto split*: *option.splits*)

lemma *inter-guards-Throw*:
$(Throw \cap_{gs} c2) = Some\ c = (c2{=}Throw \wedge c{=}Throw)$
by (*cases c2*) *auto*

lemma *inter-guards-Catch*:
$(Catch\ a1\ a2 \cap_{gs} c2) = Some\ c =$
$(\exists\, b1\ b2\ d1\ d2.\ c2{=}Catch\ b1\ b2 \wedge (a1 \cap_{gs} b1) = Some\ d1 \wedge$
$(a2 \cap_{gs} b2) = Some\ d2 \wedge c{=}Catch\ d1\ d2)$
by (*cases c2*) (*auto split*: *option.splits*)

lemmas *inter-guards-simps* = *inter-guards-Skip inter-guards-Basic inter-guards-Spec inter-guards-Seq inter-guards-Cond inter-guards-While inter-guards-Call inter-guards-DynCom inter-guards-Guard inter-guards-Throw inter-guards-Catch inter-guards-Await*

5.3.6 Subset on Guards: $c_1 \subseteq_g c_2$

consts *subseteq-guards*:: $('s, 'p, 'f, 'e)$ *com* $\times$ $('s, 'p, 'f, 'e)$ *com* $\Rightarrow$ *bool*

abbreviation
subseteq-guards-syntax :: $('s, 'p, 'f, 'e)$ *com* $\Rightarrow$ $('s, 'p, 'f, 'e)$ *com* $\Rightarrow$ *bool*
(- $\subseteq_{gs}$ - [20,20] 19)
where $c \subseteq_{gs} d$ == *subseteq-guards* (c,d)

recdef *subseteq-guards inv-image com-rel snd*
(*Skip* $\subseteq_{gs}$ *Skip*) = *True*
(*Basic f1 e1* $\subseteq_{gs}$ *Basic f2 e2*) = ((*f1*=*f2*) $\wedge$ (*e1* = *e2*))
(*Spec r1 e1* $\subseteq_{gs}$ *Spec r2 e2*) = ((*r1*=*r2*) $\wedge$ (*e1* = *e2*))
(*Seq a1 a2* $\subseteq_{gs}$ *Seq b1 b2*) = ((*a1* $\subseteq_{gs}$ *b1*) $\wedge$ (*a2* $\subseteq_{gs}$ *b2*))
(*Cond cnd1 t1 e1* $\subseteq_{gs}$ *Cond cnd2 t2 e2*) = ((*cnd1*=*cnd2*) $\wedge$ (*t1* $\subseteq_{gs}$ *t2*) $\wedge$ (*e1* $\subseteq_{gs}$ *e2*))
(*While cnd1 c1* $\subseteq_{gs}$ *While cnd2 c2*) = ((*cnd1*=*cnd2*) $\wedge$ (*c1* $\subseteq_{gs}$ *c2*))
(*Call p1* $\subseteq_{gs}$ *Call p2*) = (*p1* = *p2*)
(*DynCom P1* $\subseteq_{gs}$ *DynCom P2*) = ($\forall$ *s*. ((*P1 s*) $\subseteq_{gs}$ (*P2 s*)))
(*Guard m1 g1 c1* $\subseteq_{gs}$ *Guard m2 g2 c2*) =
((*m1*=*m2* $\wedge$ *g1*=*g2* $\wedge$ (*c1* $\subseteq_{gs}$ *c2*)) $\vee$ (*Guard m1 g1 c1* $\subseteq_{gs}$ *c2*))
(*c1* $\subseteq_{gs}$ *Guard m2 g2 c2*) = (*c1* $\subseteq_{gs}$ *c2*)
(*Await cnd1 ca1 e1* $\subseteq_{gs}$ *Await cnd2 ca2 e2*) = ((*cnd1*=*cnd2*) $\wedge$ (*ca1* $\subseteq_g$ *ca2*) $\wedge$ (*e1*=*e2*))

(*Throw* $\subseteq_{gs}$ *Throw*) = *True*
(*Catch a1 a2* $\subseteq_{gs}$ *Catch b1 b2*) = ((*a1* $\subseteq_{gs}$ *b1*) $\wedge$ (*a2* $\subseteq_{gs}$ *b2*))
(*c* $\subseteq_{gs}$ *d*) = *False*

(**hints** *cong add*: *if-cong*
recdef-wf: *wf-com-rel simp*: *com-rel.intros*)

lemma *subseteq-guards-Skip*:
$c \subseteq_{gs}$ *Skip* $\Longrightarrow$ c = *Skip*
by (*cases c*) (*auto*)

lemma *subseteq-guards-Basic*:
$c \subseteq_{gs}$ *Basic f e* $\Longrightarrow$ c = *Basic f e*
by (*cases c*) (*auto*)

lemma *subseteq-guards-Spec*:
$c \subseteq_{gs}$ *Spec r e* $\Longrightarrow c = $ *Spec r e*
by (*cases c*) (*auto*)

lemma *subseteq-guards-Seq*:
$c \subseteq_{gs}$ *Seq c1 c2* $\Longrightarrow \exists\, c1'\ c2'.\ c{=}Seq\ c1'\ c2' \wedge (c1' \subseteq_{gs} c1) \wedge (c2' \subseteq_{gs} c2)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Cond*:
$c \subseteq_{gs}$ *Cond b c1 c2* $\Longrightarrow \exists\, c1'\ c2'.\ c{=}Cond\ b\ c1'\ c2' \wedge (c1' \subseteq_{gs} c1) \wedge (c2' \subseteq_{gs} c2)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-While*:
$c \subseteq_{gs}$ *While b c'* $\Longrightarrow \exists\, c''.\ c{=}While\ b\ c'' \wedge (c'' \subseteq_{gs} c')$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Await*:
$c \subseteq_{gs}$ *Await b c' e* $\Longrightarrow \exists\, c''.\ c{=}Await\ b\ c''\ e \wedge (c'' \subseteq_{g} c')$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Call*:
$c \subseteq_{gs}$ *Call p* $\Longrightarrow c = $ *Call p*
by (*cases c*) (*auto*)

lemma *subseteq-guards-DynCom*:
$c \subseteq_{gs}$ *DynCom C* $\Longrightarrow \exists\, C'.\ c{=}DynCom\ C' \wedge (\forall s.\ C'\ s \subseteq_{gs} C\ s)$
by (*cases c*) (*auto*)

lemma *subseteq-guards-Guard*:
$c \subseteq_{gs}$ *Guard f g c'* $\Longrightarrow$
$(c \subseteq_{gs} c') \vee (\exists\, c''.\ c{=}Guard\ f\ g\ c'' \wedge (c'' \subseteq_{gs} c'))$
by (*cases c*) (*auto split*: *if-split-asm*)

lemma *subseteq-guards-Throw*:
$c \subseteq_{gs}$ *Throw* $\Longrightarrow c = $ *Throw*
by (*cases c*) (*auto*)

lemma *subseteq-guards-Catch*:
$c \subseteq_{gs}$ *Catch c1 c2* $\Longrightarrow \exists\, c1'\ c2'.\ c{=}Catch\ c1'\ c2' \wedge (c1' \subseteq_{gs} c1) \wedge (c2' \subseteq_{gs} c2)$
by (*cases c*) (*auto*)

lemmas *subseteq-guardsD* = *subseteq-guards-Skip subseteq-guards-Basic subseteq-guards-Spec subseteq-guards-Seq subseteq-guards-Cond subseteq-guards-While subseteq-guards-Call subseteq-guards-DynCom subseteq-guards-Guard subseteq-guards-Throw subseteq-guards-Catch subseteq-guards-Await*

lemma *subseteq-guards-Guard'*:
  $Guard\ f\ b\ c \subseteq_{gs} d \Longrightarrow \exists f'\ b'\ c'.\ d{=}Guard\ f'\ b'\ c'$
apply (*cases d*)
apply *auto*
done

lemma *subseteq-guards-refl*: $c \subseteq_g c$
  by (*induct c*) *auto*

end

6 Big-Step Semantics for Simpl

theory *SemanticCon* **imports** *LanguageCon EmbSimpl/Semantic* **begin**

notation
restrict-map (-|- [*90*, *91*] *90*)

definition $isAbr{::}('s,'f)\ xstate \Rightarrow bool$
  where $isAbr\ S = (\exists s.\ S{=}Abrupt\ s)$

lemma *isAbr-simps* [*simp*]:
$isAbr\ (Normal\ s) = False$
$isAbr\ (Abrupt\ s) = True$
$isAbr\ (Fault\ f) = False$
$isAbr\ Stuck = False$
by (*auto simp add*: *isAbr-def*)

lemma *isAbrE* [*consumes 1*, *elim?*]: $[\![isAbr\ S;\ \bigwedge s.\ S{=}Abrupt\ s \Longrightarrow P]\!] \Longrightarrow P$
  by (*auto simp add*: *isAbr-def*)

lemma *not-isAbrD*:
$\neg\ isAbr\ s \Longrightarrow (\exists s'.\ s{=}Normal\ s') \lor s = Stuck \lor (\exists f.\ s{=}Fault\ f)$
  by (*cases s*) *auto*

definition $isFault{::}\ ('s,'f)\ xstate \Rightarrow bool$
  where $isFault\ S = (\exists f.\ S{=}Fault\ f)$

lemma *isFault-simps* [*simp*]:
$isFault\ (Normal\ s) = False$
$isFault\ (Abrupt\ s) = False$
$isFault\ (Fault\ f) = True$
$isFault\ Stuck = False$
by (*auto simp add*: *isFault-def*)

lemma *isFaultE* [*consumes 1*, *elim?*]: $[\![isFault\ s;\ \bigwedge f.\ s{=}Fault\ f \Longrightarrow P]\!] \Longrightarrow P$
by (*auto simp add*: *isFault-def*)

lemma *not-isFault-iff*: $(\neg\ isFault\ t) = (\forall f.\ t \neq Fault\ f)$
by (*auto elim*: *isFaultE*)

6.1 Big-Step Execution: $\Gamma\vdash\langle c,\ s\rangle \Rightarrow t$

The procedure environment

type-synonym $('s,'p,'f,'e)\ body = 'p \Rightarrow ('s,'p,'f,'e)\ com\ option$

definition *no-await-body* :: $('s,'p,'f,'e)\ body \Rightarrow ('s,'p,'f)\ Semantic.body$ $(\text{-}_{\neg a}\ [98])$
where
$no\text{-}await\text{-}body\ \Gamma \equiv (\lambda x.\ case\ (\Gamma\ x)\ of\ (Some\ t) \Rightarrow if\ (noawaits\ t)\ then\ Some\ (sequential\ t)\ else\ None$
$\quad\quad | \ None \Rightarrow None$
$\quad)$

definition *parallel-env*::$('s,'p,'f)\ Semantic.body \Rightarrow ('s,'p,'f,'e')\ body$
where
$parallel\text{-}env\ \Gamma = (\lambda x.\ case\ (\Gamma\ x)\ of\ (Some\ t) \Rightarrow Some\ (parallel\ t)$
$\quad\quad | \ None \Rightarrow None)$

lemma *in-gamma-in-noawait-gamma*:
$\forall p.\ p \in dom\ (\Gamma_{\neg a}) \longrightarrow p \in\ dom\ \Gamma$
by (*simp add*: *domIff no-await-body-def option.case-eq-if*)

lemma *no-await-some-some-p*:
assumes *not-none*:$\Gamma_{\neg a}\ p = Some\ s$
shows $(\Gamma\ p) = None \Longrightarrow P$
proof −
assume $\Gamma\ p = None$
hence $None = \Gamma_{\neg a}\ p$
by (*simp add*: *no-await-body-def*)
thus *?thesis*
by (*simp add*: *not-none*)
qed

lemma *no-await-some-no-await*:
assumes *not-none*:$\Gamma_{\neg a}\ p = Some\ s \wedge (\Gamma\ p) = Some\ t$
shows *noawaits t*
proof −
have $None \neq \Gamma_{\neg a}\ p$
using *not-none* **by** *auto*

hence (*if noawaits t then Some* (*sequential t*) *else None*) $\neq$ *None*
by (*simp add*: *no-await-body-def not-none*)
thus *?thesis*
by *meson*
qed

lemma *lam1-seq*:$\Gamma 1 = \Gamma_{\neg a} \Longrightarrow \Gamma 1\ p = Some\ s \Longrightarrow \Gamma\ p = Some\ t \Longrightarrow s = sequential\ t$
unfolding *no-await-body-def*
proof –
assume *a1*: $\Gamma 1\ p = Some\ s$
assume *a2*: $\Gamma 1 = (\lambda x.\ case\ \Gamma\ x\ of\ None \Rightarrow None \mid Some\ t \Rightarrow if\ noawaits\ t\ then\ Some\ (sequential\ t)\ else\ None)$
assume $\Gamma\ p = Some\ t$
hence (*if noawaits t then Some* (*sequential t*) *else None*) $= \Gamma 1\ p$
using *a2* **by** *force*
thus *?thesis*
using *a1* **by** (*metis* (*no-types*) *option.distinct*(*2*) *option.inject*)
qed

inductive
exec::[('s,'p,'f,'e) *body*,('s,'p,'f,'e) *com*,('s,'f) *xstate*,('s,'f) *xstate*]
$\Rightarrow$ *bool* (-$\vdash_p$ $\langle$-,-$\rangle$ $\Rightarrow$ - [*60*,*20*,*98*,*98*] *89*)
for Γ::('s,'p,'f,'e) *body*
where
Skip: $\Gamma\vdash_p\langle Skip, Normal\ s\rangle \Rightarrow Normal\ s$

| *Guard*: $[\![s\in g;\ \Gamma\vdash_p\langle c, Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Guard\ f\ g\ c, Normal\ s\rangle \Rightarrow t$

| *GuardFault*: $s\notin g \Longrightarrow \Gamma\vdash_p\langle Guard\ f\ g\ c, Normal\ s\rangle \Rightarrow Fault\ f$

| *FaultProp* [*intro*,*simp*]: $\Gamma\vdash_p\langle c, Fault\ f\rangle \Rightarrow Fault\ f$

| *Basic*: $\Gamma\vdash_p\langle Basic\ f\ e, Normal\ s\rangle \Rightarrow Normal\ (f\ s)$

| *Spec*: $(s,t) \in r$
$\Longrightarrow$
$\Gamma\vdash_p\langle Spec\ r\ e, Normal\ s\rangle \Rightarrow Normal\ t$

| *SpecStuck*: $\forall t.\ (s,t) \notin r$
$\Longrightarrow$
$\Gamma\vdash_p\langle Spec\ r\ e, Normal\ s\rangle \Rightarrow Stuck$

| *Seq*: $[\![\Gamma\vdash_p\langle c_1, Normal\ s\rangle \Rightarrow s';\ \Gamma\vdash_p\langle c_2, s'\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Seq\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$

| *CondTrue*: $[\![s \in b; \Gamma \vdash_p \langle c_1, Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Cond\ b\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$

| *CondFalse*: $[\![s \notin b; \Gamma \vdash_p \langle c_2, Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Cond\ b\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$

| *WhileTrue*: $[\![s \in b; \Gamma \vdash_p \langle c, Normal\ s\rangle \Rightarrow s'; \Gamma \vdash_p \langle While\ b\ c, s'\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle While\ b\ c, Normal\ s\rangle \Rightarrow t$

| *AwaitTrue*: $[\![s \in b; \Gamma_p = \Gamma_{\neg a}\ ; \Gamma_p \vdash \langle ca, Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Await\ b\ ca\ e, Normal\ s\rangle \Rightarrow t$

| *AwaitFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Await\ b\ ca\ e, Normal\ s\rangle \Rightarrow Normal\ s$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle While\ b\ c, Normal\ s\rangle \Rightarrow Normal\ s$

| *Call*: $[\![\Gamma\ p = Some\ bdy; \Gamma \vdash_p \langle bdy, Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Call\ p, Normal\ s\rangle \Rightarrow t$

| *CallUndefined*: $[\![\Gamma\ p = None]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Call\ p, Normal\ s\rangle \Rightarrow Stuck$

| *StuckProp* [*intro*,*simp*]: $\Gamma \vdash_p \langle c, Stuck\rangle \Rightarrow Stuck$

| *DynCom*: $[\![\Gamma \vdash_p \langle (c\ s), Normal\ s\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle DynCom\ c, Normal\ s\rangle \Rightarrow t$

| *Throw*: $\Gamma \vdash_p \langle Throw, Normal\ s\rangle \Rightarrow Abrupt\ s$

| *AbruptProp* [*intro*,*simp*]: $\Gamma \vdash_p \langle c, Abrupt\ s\rangle \Rightarrow Abrupt\ s$

| *CatchMatch*: $[\![\Gamma \vdash_p \langle c_1, Normal\ s\rangle \Rightarrow Abrupt\ s'; \Gamma \vdash_p \langle c_2, Normal\ s'\rangle \Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Catch\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$
| *CatchMiss*: $[\![\Gamma \vdash_p \langle c_1, Normal\ s\rangle \Rightarrow t; \neg isAbr\ t]\!]$
$\Longrightarrow$
$\Gamma \vdash_p \langle Catch\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$

inductive-cases *exec-elim-cases* [*cases set*]:
$\Gamma\vdash_p\langle c,Fault\ f\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle c,Stuck\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle c,Abrupt\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Skip,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Guard\ f\ g\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Basic\ f\ e,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Spec\ r\ e,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Cond\ b\ c1\ c2,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle While\ b\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Await\ b\ c\ e,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Call\ p,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle DynCom\ c,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Throw,s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Catch\ c1\ c2,s\rangle \Rightarrow\ t$

inductive-cases *exec-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_p\langle c,Fault\ f\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle c,Stuck\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle c,Abrupt\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Skip,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Basic\ f\ e,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Spec\ r\ e,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle While\ b\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Await\ b\ c\ e,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle DynCom\ c,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Throw,Normal\ s\rangle \Rightarrow\ t$
$\Gamma\vdash_p\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\ t$

Relation between Concurrent Semantics and Sequential semantics

lemma *exec-seq-parallel*:
assumes *a0*:$\Gamma\vdash\langle bdy,s\rangle \Rightarrow\ t$
shows (*parallel-env* Γ)$\vdash_p$ $\langle parallel\ bdy,s\rangle \Rightarrow\ t$
using *a0*
proof(*induct*)
case (*Call p bdy s t*)
then show *?case* **by** (*simp add*: *SemanticCon.exec.Call parallel-env-def*)
next
case (*CallUndefined p s*)
then show *?case*
by (*simp add*: *SemanticCon.exec.CallUndefined parallel-env-def*)
next
case (*CatchMiss* c_1 *s t* c_2)

then show ?case
using Semantic.isAbr-def SemanticCon.exec.CatchMiss SemanticCon.isAbrE
by fastforce
qed(fastforce intro: exec.intros)+

lemma exec-block:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t;\ \Gamma\vdash_p\langle c\ s\ t,Normal\ (return\ s\ t)\rangle \Rightarrow\ u]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle \Rightarrow\ u$
apply (unfold block-def)
by (fastforce intro: exec.intros)

lemma exec-blockAbrupt:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Abrupt\ (return\ s\ t)$
apply (unfold block-def)
by (fastforce intro: exec.intros)

lemma exec-blockFault:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Fault\ f$
apply (unfold block-def)
by (fastforce intro: exec.intros)

lemma exec-blockStuck:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Stuck$
apply (unfold block-def)
by (fastforce intro: exec.intros)

lemma exec-call:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t;\ \Gamma\vdash_p\langle c\ s\ t,Normal\ (return\ s\ t)\rangle \Rightarrow\ u]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ u$
apply (simp add: call-def)
apply (rule exec-block)
apply (erule (1) Call)
apply assumption
done

lemma exec-callAbrupt:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$

$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*simp add*: *call-def*)
apply (*rule exec-blockAbrupt*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callFault*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Fault\ f$
apply (*simp add*: *call-def*)
apply (*rule exec-blockFault*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callStuck*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule exec-blockStuck*)
apply (*erule* (*1*) *Call*)
done

lemma *exec-callUndefined*:
$[\![\Gamma\ p=None]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule exec-blockStuck*)
apply (*erule CallUndefined*)
done

lemma *Fault-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\ t$ **and** *s*: $s=Fault\ f$
shows $t=Fault\ f$
using *exec s* **by** (*induct*) *auto*

lemma *Stuck-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\ t$ **and** *s*: $s=Stuck$
shows $t=Stuck$
using *exec s* **by** (*induct*) *auto*

lemma *Abrupt-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\ t$ **and** *s*: $s=Abrupt\ s'$
shows $t=Abrupt\ s'$
using *exec s* **by** (*induct*) *auto*

lemma *exec-Call-body-aux*:
$\Gamma\ p=Some\ bdy \Longrightarrow$
$\Gamma\vdash_p\langle Call\ p,s\rangle \Rightarrow t = \Gamma\vdash_p\langle bdy,s\rangle \Rightarrow t$

apply (*rule*)
apply (*fastforce elim*: *exec-elim-cases*)
apply (*cases s*)
apply (*cases t*)
apply (*auto intro*: *exec.intros dest*: *Fault-end Stuck-end Abrupt-end*)
done

lemma *exec-Call-body′*:
$p \in dom\ \Gamma \Longrightarrow$
$\Gamma\vdash_p\langle Call\ p,s\rangle \Rightarrow t = \Gamma\vdash_p\langle the\ (\Gamma\ p),s\rangle \Rightarrow t$
apply *clarsimp*
by (*rule exec-Call-body-aux*)

lemma *exec-block-Normal-elim* [*consumes 1*]:
assumes *exec-block*: $\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle \Rightarrow\ t$
assumes *Normal*:
$\bigwedge t'.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t';$
$\Gamma\vdash_p\langle c\ s\ t',Normal\ (return\ s\ t')\rangle \Rightarrow\ t]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge t'.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t';$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge f.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Stuck;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes
$[\![\Gamma\ p = None;\ t = Stuck]\!] \Longrightarrow P$
shows *P*
using *exec-block*
apply (*unfold block-def*)
apply (*elim exec-Normal-elim-cases*)
apply *simp-all*
apply (*case-tac s′*)
apply *simp-all*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*drule Abrupt-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)

apply *simp*
apply (*rule Abrupt*,*assumption*+)
apply (*drule Fault-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)
apply *simp*
apply (*drule Stuck-end*) **apply** *simp*
apply (*erule exec-Normal-elim-cases*)
apply *simp*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*, *assumption*+)
apply (*drule Fault-end*) **apply** *simp*
apply (*rule Fault*,*assumption*+)
apply (*drule Stuck-end*) **apply** *simp*
apply (*rule Stuck*,*assumption*+)
done

lemma *exec-call-Normal-elim* [*consumes 1*]:
assumes *exec-call*: $\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c, Normal\ s\rangle \Rightarrow\ t$
assumes *Normal*:
$\bigwedge bdy\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t';$
$\Gamma\vdash_p\langle c\ s\ t', Normal\ (return\ s\ t')\rangle \Rightarrow\ t]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge bdy\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t';$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge bdy\ f.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$\bigwedge bdy.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Stuck;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes *Undef*:
$[\![\Gamma\ p = None;\ t = Stuck]\!] \Longrightarrow P$
shows P
using *exec-call*
apply (*unfold call-def*)
apply (*cases* Γ *p*)
apply (*erule exec-block-Normal-elim*)
apply (*elim exec-Normal-elim-cases*)

apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Undef*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
apply (*erule exec-block-Normal-elim*)
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*,*assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt*,*assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Fault*, *assumption*+)
apply *simp*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Stuck*,*assumption*,*assumption*,*assumption*)
apply *simp*
apply (*rule Undef*,*assumption*+)
done

lemma *exec-dynCall*:
$[\![\Gamma\vdash_p\langle call\ init\ ei(p\ s)\ return\ er\ c,Normal\ s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle dynCall\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

lemma *exec-dynCall-Normal-elim*:
assumes *exec*: $\Gamma\vdash_p\langle dynCall\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle \Rightarrow\ t$
assumes *call*: $\Gamma\vdash_p\langle call\ init\ ei\ (p\ s)\ return\ er\ c,Normal\ s\rangle \Rightarrow\ t \Longrightarrow P$
shows P
using *exec*
apply (*simp add*: *dynCall-def*)
apply (*erule exec-Normal-elim-cases*)
apply (*rule call*,*assumption*)
done

lemma *exec-Call-body*:
 Γ *p=Some bdy* $\Longrightarrow$
 $\Gamma\vdash_p\langle Call\ p,s\rangle \Rightarrow\ t = \Gamma\vdash_p\langle the\ (\Gamma\ p),s\rangle \Rightarrow\ t$
apply (*rule*)
apply (*fastforce elim*: *exec-elim-cases*)
apply (*cases s*)
apply (*cases t*)
apply (*fastforce intro*: *exec.intros dest*: *Fault-end Abrupt-end Stuck-end*)+
done

lemma *exec-Seq'*: $[\![\Gamma\vdash_p\langle c1,s\rangle \Rightarrow\ s'; \Gamma\vdash_p\langle c2,s'\rangle \Rightarrow\ s'']\!]$
 $\Longrightarrow$
 $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow\ s''$
 apply (*cases s*)
 apply (*fastforce intro*: *exec.intros*)
 apply (*fastforce dest*: *Abrupt-end*)
 apply (*fastforce dest*: *Fault-end*)
 apply (*fastforce dest*: *Stuck-end*)
 done

lemma *exec-assoc*: $\Gamma\vdash_p\langle Seq\ c1\ (Seq\ c2\ c3),s\rangle \Rightarrow\ t = \Gamma\vdash_p\langle Seq\ (Seq\ c1\ c2)\ c3,s\rangle \Rightarrow\ t$
 by (*blast elim*!: *exec-elim-cases intro*: *exec-Seq'*)

6.2 Big-Step Execution with Recursion Limit: $\Gamma\vdash\langle c,\ s\rangle =n\Rightarrow t$

inductive *execn*::[('s,'p,'f,'e) *body*,('s,'p,'f,'e) *com*,('s,'f) *xstate*,*nat*,('s,'f) *xstate*]
 $\Rightarrow$ *bool* (-$\vdash_p$ ⟨-,-⟩ =-$\Rightarrow$ - [60,20,98,65,98] 89)
 for Γ::('s,'p,'f,'e) *body*
where
 Skip: $\Gamma\vdash_p\langle Skip,Normal\ s\rangle =n\Rightarrow\ Normal\ s$
| *Guard*: $[\![s\in g; \Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow\ t]\!]$
 $\Longrightarrow$
 $\Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ t$

| *GuardFault*: $s\notin g \Longrightarrow \Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ Fault\ f$

| *FaultProp* [*intro*,*simp*]: $\Gamma\vdash_p\langle c,Fault\ f\rangle =n\Rightarrow\ Fault\ f$

| *Basic*: $\Gamma\vdash_p\langle Basic\ f\ e,Normal\ s\rangle =n\Rightarrow\ Normal\ (f\ s)$

| *Spec*: $(s,t) \in r$
 $\Longrightarrow$
 $\Gamma\vdash_p\langle Spec\ r\ e,Normal\ s\rangle =n\Rightarrow\ Normal\ t$

| *SpecStuck*: $\forall\, t.\ (s,t) \notin r$
$\Longrightarrow$
$\Gamma\vdash_p\langle Spec\ r\ e, Normal\ s\rangle =n\Rightarrow\ Stuck$

| *Seq*: $[\![\Gamma\vdash_p\langle c_1, Normal\ s\rangle =n\Rightarrow\ s'; \Gamma\vdash_p\langle c_2, s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Seq\ c_1\ c_2, Normal\ s\rangle =n\Rightarrow\ t$

| *CondTrue*: $[\![s \in b; \Gamma\vdash_p\langle c_1, Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Cond\ b\ c_1\ c_2, Normal\ s\rangle =n\Rightarrow\ t$

| *CondFalse*: $[\![s \notin b; \Gamma\vdash_p\langle c_2, Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Cond\ b\ c_1\ c_2, Normal\ s\rangle =n\Rightarrow\ t$

| *WhileTrue*: $[\![s \in b; \Gamma\vdash_p\langle c, Normal\ s\rangle =n\Rightarrow\ s';$
$\Gamma\vdash_p\langle While\ b\ c, s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle While\ b\ c, Normal\ s\rangle =n\Rightarrow\ t$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle While\ b\ c, Normal\ s\rangle =n\Rightarrow\ Normal\ s$

| *AwaitTrue*: $[\![s \in b;\ \ \Gamma 1 = \Gamma_{\neg a}\ ; \Gamma 1\vdash\langle c, Normal\ s\rangle =n\Rightarrow t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ s\rangle =n\Rightarrow t$
| *AwaitFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Await\ b\ ca\ e, Normal\ s\rangle\ =n\Rightarrow\ Normal\ s$

| *Call*: $[\![\Gamma\ p = Some\ bdy; \Gamma\vdash_p\langle bdy, Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Call\ p\ , Normal\ s\rangle = Suc\ n\Rightarrow\ t$

| *CallUndefined*: $[\![\Gamma\ p = None]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Call\ p\ , Normal\ s\rangle = Suc\ n\Rightarrow\ Stuck$

| *StuckProp* [*intro*,*simp*]: $\Gamma\vdash_p\langle c, Stuck\rangle =n\Rightarrow\ Stuck$

| *DynCom*: $[\![\Gamma\vdash_p\langle(c\ s), Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle DynCom\ c, Normal\ s\rangle =n\Rightarrow\ t$

| *Throw*: $\Gamma\vdash_p\langle Throw, Normal\ s\rangle =n\Rightarrow\ Abrupt\ s$

| *AbruptProp* [*intro*,*simp*]: $\Gamma\vdash_p\langle c,Abrupt\ s\rangle =n\Rightarrow\ Abrupt\ s$

| *CatchMatch*: $[\![\Gamma\vdash_p\langle c_1,Normal\ s\rangle =n\Rightarrow\ Abrupt\ s'; \Gamma\vdash_p\langle c_2,Normal\ s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Catch\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$
| *CatchMiss*: $[\![\Gamma\vdash_p\langle c_1,Normal\ s\rangle =n\Rightarrow\ t; \neg isAbr\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Catch\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$

inductive-cases *execn-elim-cases* [*cases set*]:
$\Gamma\vdash_p\langle c,Fault\ f\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle c,Stuck\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle c,Abrupt\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Skip,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Guard\ f\ g\ c,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Basic\ f\ e,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Spec\ r\ e,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Cond\ b\ c1\ c2,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle While\ b\ c,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Await\ b\ c\ e,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Call\ p\ ,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle DynCom\ c,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Throw,s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Catch\ c1\ c2,s\rangle =n\Rightarrow\ t$

inductive-cases *execn-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_p\langle c,Fault\ f\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle c,Stuck\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle c,Abrupt\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Skip,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Basic\ f\ e,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Spec\ r\ e,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Seq\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Cond\ b\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle While\ b\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Await\ b\ c\ e,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle DynCom\ c,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Throw,Normal\ s\rangle =n\Rightarrow\ t$
$\Gamma\vdash_p\langle Catch\ c1\ c2,Normal\ s\rangle =n\Rightarrow\ t$

lemma *execn-Skip'*: $\Gamma\vdash_p\langle Skip,t\rangle =n\Rightarrow t$
by (*cases t*) (*auto intro*: *execn.intros*)

lemma *execn-Fault-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: *s=Fault f*
shows *t=Fault f*

using *exec s* **by** (*induct*) *auto*

lemma *execn-Stuck-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: *s=Stuck*
shows *t=Stuck*
using *exec s* **by** (*induct*) *auto*

lemma *execn-Abrupt-end*: **assumes** *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$ **and** *s*: *s=Abrupt s'*
shows *t=Abrupt s'*
using *exec s* **by** (*induct*) *auto*

lemma *execn-block*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t;\ \Gamma\vdash_p\langle c\ s\ t,Normal\ (return\ s\ t)\rangle =n\Rightarrow u]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ u$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockAbrupt*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockFault*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ Fault\ f$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-blockStuck*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ Stuck$
apply (*unfold block-def*)
by (*fastforce intro*: *execn.intros*)

lemma *execn-call*:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t;$
$\Gamma\vdash_p\langle c\ s\ t,Normal\ (return\ s\ t)\rangle =Suc\ n\Rightarrow\ u]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ u$
apply (*simp add*: *call-def*)
apply (*rule execn-block*)
apply (*erule* (*1*) *Call*)
apply *assumption*

done

lemma *execn-callAbrupt*:
$[\![\Gamma\ p=Some\ bdy;\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Abrupt\ (return\ s\ t)$
apply (*simp add*: *call-def*)
apply (*rule execn-blockAbrupt*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callFault*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Fault\ f$
apply (*simp add*: *call-def*)
apply (*rule execn-blockFault*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callStuck*:
$[\![\Gamma\ p=Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule execn-blockStuck*)
apply (*erule* (*1*) *Call*)
done

lemma *execn-callUndefined*:
$[\![\Gamma\ p=None]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =Suc\ n\Rightarrow\ Stuck$
apply (*simp add*: *call-def*)
apply (*rule execn-blockStuck*)
apply (*erule CallUndefined*)
done

lemma *execn-block-Normal-elim* [*consumes 1*]:
assumes *execn-block*: $\Gamma\vdash_p\langle block\ init\ ei\ bdy\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t$
assumes *Normal*:
$\bigwedge t'.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Normal\ t';$
$\Gamma\vdash_p\langle c\ s\ t',Normal\ (return\ s\ t')\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge t'.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t';$

$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge f.$
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Fault\ f;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$[\![\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =n\Rightarrow\ Stuck;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes *Undef*:
$[\![\Gamma\ p = None;\ t = Stuck]\!] \Longrightarrow P$
shows P
using *execn-block*
apply (*unfold block-def*)
apply (*elim execn-Normal-elim-cases*)
apply *simp-all*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*drule execn-Abrupt-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt,assumption+*)
apply (*drule execn-Fault-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*drule execn-Stuck-end*) **apply** *simp*
apply (*erule execn-Normal-elim-cases*)
apply *simp*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Normal,assumption+*)
apply (*drule execn-Fault-end*) **apply** *simp*
apply (*rule Fault,assumption+*)
apply (*drule execn-Stuck-end*) **apply** *simp*
apply (*rule Stuck,assumption+*)
done

lemma *execn-call-Normal-elim* [*consumes 1*]:
assumes *exec-call*: $\Gamma\vdash_p\langle call\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t$
assumes *Normal*:
$\bigwedge bdy\ i\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle =i\Rightarrow\ Normal\ t';$
$\Gamma\vdash_p\langle c\ s\ t',Normal\ (return\ s\ t')\rangle =Suc\ i\Rightarrow\ t;\ n = Suc\ i]\!]$

$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge bdy\ i\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle\ =i\Rightarrow\ \ Abrupt\ t';\ n = Suc\ i;$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge bdy\ i\ f.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle\ =i\Rightarrow\ \ Fault\ f;\ n = Suc\ i;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$\bigwedge bdy\ i.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle\ =i\Rightarrow\ \ Stuck;\ n = Suc\ i;$
$t = Stuck]\!]$
$\Longrightarrow P$
assumes *Undef*:
$\bigwedge i.\ [\![\Gamma\ p = None;\ n = Suc\ i;\ t = Stuck]\!] \Longrightarrow P$
shows *P*
using *exec-call*
apply (*unfold call-def*)
apply (*cases n*)
apply (*simp only*: *block-def*)
apply (*fastforce elim*: *execn-Normal-elim-cases*)
apply (*cases* Γ *p*)
apply (*erule execn-block-Normal-elim*)
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Undef*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
apply (*erule execn-block-Normal-elim*)
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Abrupt*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)

apply *simp*
apply (*rule Fault*,*assumption*+)
apply *simp*
apply (*elim execn-Normal-elim-cases*)
apply *simp*
apply (*rule Stuck*,*assumption*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*,*assumption*,*assumption*)
apply (*rule Undef*,*assumption*+)
done

lemma *execn-dynCall*:
$[\![\Gamma\vdash_p\langle call\ init\ ei\ (p\ s)\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle dynCall\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

lemma *execn-dynCall-Normal-elim*:
assumes *exec*: $\Gamma\vdash_p\langle dynCall\ init\ ei\ p\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t$
assumes $\Gamma\vdash_p\langle call\ init\ ei\ (p\ s)\ return\ er\ c,Normal\ s\rangle =n\Rightarrow\ t \Longrightarrow P$
shows P
using *exec*
apply (*simp add*: *dynCall-def*)
apply (*erule execn-Normal-elim-cases*)
apply *fact*
done

lemma *execn-Seq'*:
$[\![\Gamma\vdash_p\langle c1,s\rangle =n\Rightarrow\ s';\ \Gamma\vdash_p\langle c2,s'\rangle =n\Rightarrow\ s'']\!]$
$\Longrightarrow$
$\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle =n\Rightarrow\ s''$
apply (*cases s*)
apply (*fastforce intro*: *execn.intros*)
apply (*fastforce dest*: *execn-Abrupt-end*)
apply (*fastforce dest*: *execn-Fault-end*)
apply (*fastforce dest*: *execn-Stuck-end*)
done
thm *execn.intros*
lemma *execn-mono*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$
shows $\bigwedge m.\ n \leq m \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =m\Rightarrow\ t$
using *exec*
by (*induct*)(*auto intro*: *execn.intros Semantic.execn-mono dest*: *Suc-le-D*)

lemma *execn-Suc*:
$\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =Suc\ n\Rightarrow\ t$
by (*rule execn-mono* [*OF - le-refl* [*THEN le-SucI*]])

lemma *execn-assoc*:
$\Gamma\vdash_p\langle Seq\ c1\ (Seq\ c2\ c3),s\rangle =n\Rightarrow\ t = \Gamma\vdash_p\langle Seq\ (Seq\ c1\ c2)\ c3,s\rangle =n\Rightarrow\ t$
by (*auto elim*!: *execn-elim-cases intro*: *execn-Seq'*)

lemma *execn-to-exec*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$
shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
using *execn*
by (*induct*)(*auto intro*: *exec.intros Semantic.execn-to-exec*)

lemma *exec-to-execn*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
shows $\exists\, n.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\ t$
using *execn*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Guard* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Spec* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *SpecStuck* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*Seq c1 s s' c2 s''*)
then obtain *n m* **where**
$\Gamma\vdash_p\langle c1,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash_p\langle c2,s'\rangle =m\Rightarrow\ s''$
by *blast*
then have
$\Gamma\vdash_p\langle c1,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$
$\Gamma\vdash_p\langle c2,s'\rangle =max\ n\ m\Rightarrow\ s''$
by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
thus *?case*
by (*iprover intro*: *execn.intros*)
next
case *CondTrue* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *CondFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*WhileTrue s b c s' s''*)
then obtain *n m* **where**
$\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash_p\langle While\ b\ c,s'\rangle =m\Rightarrow\ s''$
by *blast*

then have
$\Gamma\vdash_p\langle c,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$ $\Gamma\vdash_p\langle While\ b\ c,s'\rangle =max\ n\ m\Rightarrow\ s''$
by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
with *WhileTrue*
show *?case*
by (*iprover intro*: *execn.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Call* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *StuckProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *DynCom* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Throw* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *AbruptProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*CatchMatch c1 s s' c2 s''*)
then obtain *n m* **where**
$\Gamma\vdash_p\langle c1,Normal\ s\rangle =n\Rightarrow\ Abrupt\ s'$ $\Gamma\vdash_p\langle c2,Normal\ s'\rangle =m\Rightarrow\ s''$
by *blast*
then have
$\Gamma\vdash_p\langle c1,Normal\ s\rangle =max\ n\ m\Rightarrow\ Abrupt\ s'$
$\Gamma\vdash_p\langle c2,Normal\ s'\rangle =max\ n\ m\Rightarrow\ s''$
by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
with *CatchMatch.hyps* **show** *?case*
by (*iprover intro*: *execn.intros*)
next
case *CatchMiss* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*AwaitTrue s b c t*) **thus** *?case* **by** (*meson exec-to-execn execn.intros*)
next
case (*AwaitFalse s b ca*) **thus** *?case* **by** (*meson exec-to-execn execn.intros*)
qed

theorem *exec-iff-execn*: $(\Gamma\vdash_p\langle c,s\rangle \Rightarrow t) = (\exists\, n.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t)$
by (*iprover intro*: *exec-to-execn execn-to-exec*)

definition *nfinal-notin*:: $('s,'p,'f,'e)$ *body* $\Rightarrow$ $('s,'p,'f,'e)$ *com* $\Rightarrow$ $('s,'f)$ *xstate* $\Rightarrow$ *nat*
$\Rightarrow$ $('s,'f)$ *xstate set* $\Rightarrow$ *bool*
$(\text{-}\vdash_p\ \langle\text{-},\text{-}\rangle =\text{-}\Rightarrow\notin\text{-}\ \ [60,20,98,65,60]\ 89)$ **where**
$\Gamma\vdash_p\ \langle c,s\rangle =n\Rightarrow\notin T = (\forall\, t.\ \Gamma\vdash_p\ \langle c,s\rangle =n\Rightarrow t \longrightarrow t\notin T)$

definition *final-notin*:: $('s,'p,'f,'e)$ *body* $\Rightarrow$ $('s,'p,'f,'e)$ *com* $\Rightarrow$ $('s,'f)$ *xstate*
$\Rightarrow$ $('s,'f)$ *xstate set* $\Rightarrow$ *bool*
$(\text{-}\vdash_p \langle\text{-},\text{-}\rangle \Rightarrow\notin\text{-}\ [60,20,98,60]\ 89)$ **where**
$\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin T = (\forall t.\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t \longrightarrow t\notin T)$

lemma *final-notinI*: $[\![\bigwedge t.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t \Longrightarrow t \notin T]\!] \Longrightarrow \Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin T$
by (*simp add*: *final-notin-def*)

lemma *noFaultStuck-Call-body'*: $p \in dom\ \Gamma \Longrightarrow$
$\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) =$
$\Gamma\vdash_p\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*clarsimp simp add*: *final-notin-def exec-Call-body*)

lemma *noFault-startn*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **and** *t*: $t\neq Fault\ f$
shows $s\neq Fault\ f$
using *execn t* **by** (*induct*) *auto*

lemma *noFault-start*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$ **and** *t*: $t\neq Fault\ f$
shows $s\neq Fault\ f$
using *exec t* **by** (*induct*) *auto*

lemma *noStuck-startn*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **and** *t*: $t\neq Stuck$
shows $s\neq Stuck$
using *execn t* **by** (*induct*) *auto*

lemma *noStuck-start*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$ **and** *t*: $t\neq Stuck$
shows $s\neq Stuck$
using *exec t* **by** (*induct*) *auto*

lemma *noAbrupt-startn*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **and** *t*: $\forall t'.\ t\neq Abrupt\ t'$
shows $s\neq Abrupt\ s'$
using *execn t* **by** (*induct*) *auto*

lemma *noAbrupt-start*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$ **and** *t*: $\forall t'.\ t\neq Abrupt\ t'$
shows $s\neq Abrupt\ s'$
using *exec t* **by** (*induct*) *auto*

lemma *noFaultn-startD*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow Normal\ t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-startn*)

lemma *noFaultn-startD'*: $t\neq Fault\ f \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t \Longrightarrow s \neq Fault\ f$
by (*auto dest*: *noFault-startn*)

lemma *noFault-startD*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow$ *Normal t* $\Longrightarrow s \neq$ *Fault f*
by (*auto dest*: *noFault-start*)

lemma *noFault-startD'*: $t\neq$*Fault f*$\Longrightarrow \Gamma\vdash_p\langle c,s\rangle \Rightarrow t \Longrightarrow s \neq$ *Fault f*
by (*auto dest*: *noFault-start*)

lemma *noStuckn-startD*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$ *Normal t* $\Longrightarrow s \neq$ *Stuck*
by (*auto dest*: *noStuck-startn*)

lemma *noStuckn-startD'*: $t\neq$*Stuck* $\Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t \Longrightarrow s \neq$ *Stuck*
by (*auto dest*: *noStuck-startn*)

lemma *noStuck-startD*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow$ *Normal t* $\Longrightarrow s \neq$ *Stuck*
by (*auto dest*: *noStuck-start*)

lemma *noStuck-startD'*: $t\neq$*Stuck* $\Longrightarrow \Gamma\vdash_p\langle c,s\rangle \Rightarrow t \Longrightarrow s \neq$ *Stuck*
by (*auto dest*: *noStuck-start*)

lemma *noAbruptn-startD*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$ *Normal t* $\Longrightarrow s \neq$ *Abrupt s'*
by (*auto dest*: *noAbrupt-startn*)

lemma *noAbrupt-startD*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow$ *Normal t* $\Longrightarrow s \neq$ *Abrupt s'*
by (*auto dest*: *noAbrupt-start*)

lemma *noFaultnI*: $[\![\bigwedge t.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t \Longrightarrow t\neq$*Fault f*$]\!] \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{$*Fault f*$\}$
by (*simp add*: *nfinal-notin-def*)

lemma *noFaultnI'*:
assumes *contr*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$ *Fault f* $\Longrightarrow$ *False*
shows $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{$*Fault f*$\}$
proof (*rule noFaultnI*)
fix *t* **assume** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
with *contr* **show** $t \neq$ *Fault f*
by (*cases t=Fault f*) *auto*
qed

lemma *noFaultn-def'*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{$*Fault f*$\} = (\neg\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$ *Fault f*$)$
apply *rule*
apply (*fastforce simp add*: *nfinal-notin-def*)
apply (*fastforce intro*: *noFaultnI'*)
done

lemma *noStucknI*: $[\![\bigwedge t.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t \Longrightarrow t\neq$*Stuck*$]\!] \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{$*Stuck*$\}$

by (*simp add*: *nfinal-notin-def*)

lemma *noStucknI'*:

assumes *contr*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$ *Stuck* $\Longrightarrow$ *False*
shows $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{Stuck\}$
proof (*rule noStucknI*)
fix t **assume** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
with *contr* **show** $t \neq Stuck$
by (*cases t*) *auto*
qed

lemma *noStuckn-def'*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{Stuck\} = (\neg\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow Stuck)$
apply *rule*
apply (*fastforce simp add*: *nfinal-notin-def*)
apply (*fastforce intro*: *noStucknI'*)
done

lemma *noFaultI*: $[\![\bigwedge t.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq Fault\ f]\!] \Longrightarrow\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Fault\ f\}$
by (*simp add*: *final-notin-def*)

lemma *noFaultI'*:
assumes *contr*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow Fault\ f \Longrightarrow False$
shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Fault\ f\}$
proof (*rule noFaultI*)
fix t **assume** $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
with *contr* **show** $t \neq Fault\ f$
by (*cases t=Fault f*) *auto*
qed

lemma *noFaultE*:
$[\![\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow Fault\ f]\!] \Longrightarrow P$
by (*auto simp add*: *final-notin-def*)

lemma *noFault-def'*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Fault\ f\} = (\neg\Gamma\vdash_p\langle c,s\rangle \Rightarrow Fault\ f)$
apply *rule*
apply (*fastforce simp add*: *final-notin-def*)
apply (*fastforce intro*: *noFaultI'*)
done

lemma *noStuckI*: $[\![\bigwedge t.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq Stuck]\!] \Longrightarrow\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Stuck\}$
by (*simp add*: *final-notin-def*)

lemma *noStuckI'*:
assumes *contr*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow Stuck \Longrightarrow False$
shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Stuck\}$
proof (*rule noStuckI*)
fix t **assume** $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
with *contr* **show** $t \neq Stuck$
by (*cases t*) *auto*
qed

lemma *noStuckE*:
$[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow Stuck]\!] \Longrightarrow P$
by (*auto simp add*: *final-notin-def*)

lemma *noStuck-def'*: $\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Stuck\} = (\neg\Gamma\vdash_p \langle c,s\rangle \Rightarrow Stuck)$
apply *rule*
apply (*fastforce simp add*: *final-notin-def*)
apply (*fastforce intro*: *noStuckI'*)
done

lemma *noFaultn-execD*: $[\![\Gamma\vdash_p \langle c,s\rangle =n\Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash_p \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow t\neq Fault$
f
by (*simp add*: *nfinal-notin-def*)

lemma *noFault-execD*: $[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t]\!] \Longrightarrow t\neq Fault\ f$
by (*simp add*: *final-notin-def*)

lemma *noFaultn-exec-startD*: $[\![\Gamma\vdash_p \langle c,s\rangle =n\Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash_p \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow$
$s\neq Fault\ f$
by (*auto simp add*: *nfinal-notin-def dest*: *noFaultn-startD*)

lemma *noFault-exec-startD*: $[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Fault\ f\};\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t]\!] \Longrightarrow s\neq Fault\ f$
by (*auto simp add*: *final-notin-def dest*: *noFault-startD*)

lemma *noStuckn-execD*: $[\![\Gamma\vdash_p \langle c,s\rangle =n\Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow t\neq Stuck$
by (*simp add*: *nfinal-notin-def*)

lemma *noStuck-execD*: $[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t]\!] \Longrightarrow t\neq Stuck$
by (*simp add*: *final-notin-def*)

lemma *noStuckn-exec-startD*: $[\![\Gamma\vdash_p \langle c,s\rangle =n\Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow s\neq Stuck$
by (*auto simp add*: *nfinal-notin-def dest*: *noStuckn-startD*)

lemma *noStuck-exec-startD*: $[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t]\!] \Longrightarrow s\neq Stuck$
by (*auto simp add*: *final-notin-def dest*: *noStuck-startD*)

lemma *noFaultStuckn-execD*:
$[\![\Gamma\vdash_p \langle c,s\rangle =n\Rightarrow\notin\{Fault\ True,Fault\ False,Stuck\};\ \Gamma\vdash_p \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow$
$t\notin\{Fault\ True,Fault\ False,Stuck\}$
by (*simp add*: *nfinal-notin-def*)

lemma *noFaultStuck-execD*: $[\![\Gamma\vdash_p \langle c,s\rangle \Rightarrow\notin\{Fault\ True,Fault\ False,Stuck\};\ \Gamma\vdash_p \langle c,s\rangle$
$\Rightarrow t]\!]$
$\Longrightarrow t\notin\{Fault\ True,Fault\ False,Stuck\}$
by (*simp add*: *final-notin-def*)

lemma *noFaultStuckn-exec-startD*:

$[\![\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin\{Fault\ True,\ Fault\ False,Stuck\};\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t]\!]$
$\Longrightarrow s\notin\{Fault\ True,Fault\ False,Stuck\}$
by (*auto simp add*: *nfinal-notin-def*)

lemma *noFaultStuck-exec-startD*:
$[\![\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin\{Fault\ True,\ Fault\ False,Stuck\};\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t]\!]$
$\Longrightarrow s\notin\{Fault\ True,Fault\ False,Stuck\}$
by (*auto simp add*: *final-notin-def*)

lemma *noStuck-Call*:
assumes *noStuck*: $\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
shows $p \in dom\ \Gamma$
proof (*cases* $p \in dom\ \Gamma$)
case *True* **thus** *?thesis* **by** *simp*
next
case *False*
hence $\Gamma\ p = None$ **by** *auto*
hence $\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle \Rightarrow Stuck$
by (*rule exec.CallUndefined*)
with *noStuck* **show** *?thesis*
by (*auto simp add*: *final-notin-def*)
qed

lemma *Guard-noFaultStuckD*:
assumes $\Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
assumes $f \notin F$
shows $s \in g$
using *assms*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)

lemma *final-notin-to-finaln*:
assumes *notin*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin T$
shows $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin T$
proof (*clarsimp simp add*: *nfinal-notin-def*)
fix t **assume** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **and** $t\in T$
with *notin* **show** *False*
by (*auto intro*: *execn-to-exec simp add*: *final-notin-def*)
qed

lemma *noFault-Call-body*:
$\Gamma\ p=Some\ bdy\Longrightarrow$
$\Gamma\vdash_p\langle Call\ p\ ,Normal\ s\rangle \Rightarrow\notin\{Fault\ f\} =$
$\Gamma\vdash_p\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin\{Fault\ f\}$
by (*simp add*: *noFault-def′ exec-Call-body*)

lemma *noStuck-Call-body*:
$\Gamma\ p=Some\ bdy\Longrightarrow$

$\Gamma\vdash_p\langle Call\ p,Normal\ s\rangle \Rightarrow\notin\{Stuck\} =$
$\Gamma\vdash_p\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*simp add*: *noStuck-def′ exec-Call-body*)

lemma *exec-final-notin-to-execn*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin T \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin T$
by (*auto simp add*: *final-notin-def nfinal-notin-def dest*: *execn-to-exec*)

lemma *execn-final-notin-to-exec*: $\forall n.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin T \Longrightarrow \Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin T$
by (*auto simp add*: *final-notin-def nfinal-notin-def dest*: *exec-to-execn*)

lemma *exec-final-notin-iff-execn*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow\notin T = (\forall n.\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow\notin T)$
by (*auto intro*: *exec-final-notin-to-execn execn-final-notin-to-exec*)

lemma *Seq-NoFaultStuckD2*:
assumes *noabort*: $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
shows $\forall t.\ \Gamma\vdash_p\langle c1,s\rangle \Rightarrow t \longrightarrow t\notin (\{Stuck\} \cup Fault\ `\ F) \longrightarrow$
$\Gamma\vdash_p\langle c2,t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
using *noabort*
by (*auto simp add*: *final-notin-def intro*: *exec-Seq′*) **lemma** *Seq-NoFaultStuckD1*:
assumes *noabort*: $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
shows $\Gamma\vdash_p\langle c1,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
proof (*rule final-notinI*)
fix *t*
assume *exec-c1*: $\Gamma\vdash_p\langle c1,s\rangle \Rightarrow t$
show $t \notin \{Stuck\} \cup Fault\ `\ F$
proof
assume $t \in \{Stuck\} \cup Fault\ `\ F$
moreover
{
assume $t = Stuck$
with *exec-c1*
have $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow Stuck$
by (*auto intro*: *exec-Seq′*)
with *noabort* **have** *False*
by (*auto simp add*: *final-notin-def*)
hence *False* **..**
}
moreover
{
assume $t \in Fault\ `\ F$
then obtain *f* **where**
t: *t=Fault f* **and** *f*: $f \in F$
by *auto*
from *t exec-c1*
have $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow Fault\ f$
by (*auto intro*: *exec-Seq′*)
with *noabort f* **have** *False*
by (*auto simp add*: *final-notin-def*)
hence *False* **..**

}
ultimately show *False* **by** *auto*
qed
qed

lemma *Seq-NoFaultStuckD2′*:
assumes *noabort*: $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
shows $\forall\, t.\ \Gamma\vdash_p\langle c1,s\rangle \Rightarrow t \longrightarrow t\notin (\{Stuck\} \cup Fault\ `\ F) \longrightarrow$
$\Gamma\vdash_p\langle c2,t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
using *noabort*
by (*auto simp add*: *final-notin-def intro*: *exec-Seq′*)

6.3 Lemmas about *LanguageCon.sequence*, *LanguageCon.flatten* and *LanguageCon.normalize*

lemma *execn-sequence-app*: $\bigwedge s\ s'\ t.$
$[\![\Gamma\vdash_p\langle sequence\ Seq\ xs,Normal\ s\rangle =n\Rightarrow s'; \Gamma\vdash_p\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow t]\!]$
$\Longrightarrow \Gamma\vdash_p\langle sequence\ Seq\ (xs@ys),Normal\ s\rangle =n\Rightarrow t$
proof (*induct xs*)
case *Nil*
thus *?case* **by** (*auto elim*: *execn-Normal-elim-cases*)
next
case (*Cons x xs*)
have *exec-x-xs*: $\Gamma\vdash_p\langle sequence\ Seq\ (x\ \#\ xs),Normal\ s\rangle =n\Rightarrow s'$ **by** *fact*
have *exec-ys*: $\Gamma\vdash_p\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases xs*)
case *Nil*
with *exec-x-xs* **have** $\Gamma\vdash_p\langle x,Normal\ s\rangle =n\Rightarrow s'$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Nil exec-ys* **show** *?thesis*
by (*cases ys*) (*auto intro*: *execn.intros elim*: *execn-elim-cases*)
next
case *Cons*
with *exec-x-xs*
obtain s'' **where**
exec-x: $\Gamma\vdash_p\langle x,Normal\ s\rangle =n\Rightarrow s''$ **and**
exec-xs: $\Gamma\vdash_p\langle sequence\ Seq\ xs,s''\rangle =n\Rightarrow s'$
by (*auto elim*: *execn-Normal-elim-cases*)
show *?thesis*
proof (*cases* s'')
case (*Normal* s''')
from *Cons.hyps* [*OF exec-xs* [*simplified Normal*] *exec-ys*]
have $\Gamma\vdash_p\langle sequence\ Seq\ (xs\ @\ ys),Normal\ s'''\rangle =n\Rightarrow t$ **.**
with *Cons exec-x Normal*
show *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Abrupt* s''')

with *exec-xs* **have** s'=*Abrupt* s'''
by (*auto dest*: *execn-Abrupt-end*)
with *exec-ys* **have** t=*Abrupt* s'''
by (*auto dest*: *execn-Abrupt-end*)
with *exec-x Abrupt Cons* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Fault f*)
with *exec-xs* **have** s'=*Fault f*
by (*auto dest*: *execn-Fault-end*)
with *exec-ys* **have** t=*Fault f*
by (*auto dest*: *execn-Fault-end*)
with *exec-x Fault Cons* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-xs* **have** s'=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *exec-ys* **have** t=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *exec-x Stuck Cons* **show** *?thesis*
by (*auto intro*: *execn.intros*)
qed
qed
qed

lemma *execn-sequence-appD*: $\bigwedge s\ t.\ \Gamma\vdash_p\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ =n⇒ $t \Longrightarrow$
$\exists s'.\ \Gamma\vdash_p\langle$*sequence Seq xs*,*Normal s*$\rangle$ =n⇒ $s' \wedge \Gamma\vdash_p\langle$*sequence Seq ys*,$s'\rangle$ =n⇒ t
proof (*induct xs*)
case *Nil*
thus *?case*
by (*auto intro*: *execn.intros*)
next
case (*Cons x xs*)
have *exec-app*: $\Gamma\vdash_p\langle$*sequence Seq* ((*x* # *xs*) @ *ys*),*Normal s*$\rangle$ =n⇒ t **by** *fact*
show *?case*
proof (*cases xs*)
case *Nil*
with *exec-app* **show** *?thesis*
by (*cases ys*) (*auto elim*: *execn-Normal-elim-cases intro*: *execn-Skip'*)
next
case *Cons*
with *exec-app* **obtain** s' **where**
exec-x: $\Gamma\vdash_p\langle x$,*Normal s*$\rangle$ =n⇒ s' **and**
exec-xs-ys: $\Gamma\vdash_p\langle$*sequence Seq* (*xs* @ *ys*),$s'\rangle$ =n⇒ t
by (*auto elim*: *execn-Normal-elim-cases*)
show *?thesis*

```
    proof (cases s')
      case (Normal s'')
      from Cons.hyps [OF exec-xs-ys [simplified Normal]] Normal exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case (Abrupt s'')
      with exec-xs-ys have t=Abrupt s''
        by (auto dest: execn-Abrupt-end)
      with Abrupt exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case (Fault f)
      with exec-xs-ys have t=Fault f
        by (auto dest: execn-Fault-end)
      with Fault exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    next
      case Stuck
      with exec-xs-ys have t=Stuck
        by (auto dest: execn-Stuck-end)
      with Stuck exec-x Cons
      show ?thesis
        by (auto intro: execn.intros)
    qed
  qed
qed

lemma execn-sequence-appE [consumes 1]:
  ⟦Γ⊢p⟨sequence Seq (xs @ ys),Normal s⟩ =n⇒ t;
   ⋀s'. ⟦Γ⊢p⟨sequence Seq xs,Normal s⟩ =n⇒ s';Γ⊢p⟨sequence Seq ys,s'⟩ =n⇒ t⟧
⟹ P
   ⟧ ⟹ P
  by (auto dest: execn-sequence-appD)

lemma execn-to-execn-sequence-flatten:
  assumes exec: Γ⊢p⟨c,s⟩ =n⇒ t
  shows Γ⊢p⟨sequence Seq (flatten c),s⟩ =n⇒ t
using exec
proof induct
  case (Seq c1 c2 n s s' s'') thus ?case
    by (auto intro: execn.intros execn-sequence-app)
qed (auto intro: execn.intros)

lemma execn-to-execn-normalize:
  assumes exec: Γ⊢p⟨c,s⟩ =n⇒ t
  shows Γ⊢p⟨normalize c,s⟩ =n⇒ t
```

using *exec*
proof *induct*
  case (*Seq c1 c2 n s s' s''*) **thus** *?case*
    by (*auto intro*: *execn-to-execn-sequence-flatten execn-sequence-app*)
next
  case (*AwaitFalse s b c n*) **thus** *?case* **using** *execn-to-execn-normalize*
    by (*simp add*: *execn.AwaitFalse*)
qed (*auto intro*: *execn.intros execn-to-execn-normalize*)

lemma *execn-sequence-flatten-to-execn*:
  shows $\bigwedge s\ t.\ \Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c),s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
proof (*induct c*)
  case (*Seq c1 c2*)
  have *exec-seq*: $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ (Seq\ c1\ c2)),s\rangle =n\Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases s*)
    case (*Normal s'*)
    with *exec-seq* **obtain** s'' **where**
      $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c1),Normal\ s'\rangle =n\Rightarrow s''$ **and**
      $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c2),s''\rangle =n\Rightarrow t$
      by (*auto elim*: *execn-sequence-appE*)
    with *Seq.hyps Normal*
    show *?thesis*
      by (*fastforce intro*: *execn.intros*)
  next
    case *Abrupt*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Abrupt-end*)
  next
    case *Fault*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Fault-end*)
  next
    case *Stuck*
    with *exec-seq*
    show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Stuck-end*)
  qed
qed *auto*

lemma *execn-normalize-to-execn*:
  shows $\bigwedge s\ t\ n.\ \Gamma\vdash_p\langle normalize\ c,s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
proof (*induct c*)
  case *Skip* **thus** *?case* **by** *simp*
next
  case *Basic* **thus** *?case* **by** *simp*
next

case *Spec* **thus** *?case* **by** *simp*
next
case (*Seq c1 c2*)
have $\Gamma\vdash_p\langle$*normalize* (*Seq c1 c2*),*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
hence *exec-norm-seq*:
$\Gamma\vdash_p\langle$*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*)),*s*$\rangle$ =*n*$\Rightarrow$ *t*
by *simp*
show *?case*
proof (*cases s*)
case (*Normal s'*)
with *exec-norm-seq* **obtain** *s''* **where**
exec-norm-c1: $\Gamma\vdash_p\langle$*sequence Seq* (*flatten* (*normalize c1*)),*Normal s'*$\rangle$ =*n*$\Rightarrow$ *s''*
and
exec-norm-c2: $\Gamma\vdash_p\langle$*sequence Seq* (*flatten* (*normalize c2*)),*s''*$\rangle$ =*n*$\Rightarrow$ *t*
by (*auto elim*: *execn-sequence-appE*)
from *execn-sequence-flatten-to-execn* [*OF exec-norm-c1*]
execn-sequence-flatten-to-execn [*OF exec-norm-c2*] *Seq.hyps Normal*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
next
case (*Abrupt s'*)
with *exec-norm-seq* **have** *t=Abrupt s'*
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Fault f*)
with *exec-norm-seq* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-norm-seq* **have** *t=Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by (*auto intro*: *execn.intros*)
qed
next
case *Cond* **thus** *?case*
by (*auto intro*: *execn.intros elim*!: *execn-elim-cases*)
next
case (*While b c*)
have $\Gamma\vdash_p\langle$*normalize* (*While b c*),*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
hence *exec-norm-w*: $\Gamma\vdash_p\langle$*While b* (*normalize c*),*s*$\rangle$ =*n*$\Rightarrow$ *t*
by *simp*
{
fix *s t w*
assume *exec-w*: $\Gamma\vdash_p\langle w,s\rangle$ =*n*$\Rightarrow$ *t*

have w=*While* b (*normalize* c) $\Longrightarrow$ $\Gamma\vdash_p\langle$*While* b c,$s\rangle$ =$n\Rightarrow$ t
using *exec-w*
proof (*induct*)
case (*WhileTrue* s b' c' n w t)
from *WhileTrue* **obtain**
s-in-b: $s \in b$ **and**
exec-c: $\Gamma\vdash_p\langle$*normalize* c,*Normal* $s\rangle$ =$n\Rightarrow$ w **and**
hyp-w: $\Gamma\vdash_p\langle$*While* b c,$w\rangle$ =$n\Rightarrow$ t
by *simp*
from *While.hyps* [*OF* *exec-c*]
have $\Gamma\vdash_p\langle c$,*Normal* $s\rangle$ =$n\Rightarrow$ w
by *simp*
with *hyp-w* *s-in-b*
have $\Gamma\vdash_p\langle$*While* b c,*Normal* $s\rangle$ =$n\Rightarrow$ t
by (*auto intro*: *execn.intros*)
with *WhileTrue* **show** *?case* **by** *simp*
qed (*auto intro*: *execn.intros*)
}
from *this* [*OF* *exec-norm-w*]
show *?case*
by *simp*
next
case *Call* **thus** *?case* **by** *simp*
next
case *DynCom* **thus** *?case* **by** (*auto intro*: *execn.intros* *elim*!: *execn-elim-cases*)
next
case *Guard* **thus** *?case* **by** (*auto intro*: *execn.intros* *elim*!: *execn-elim-cases*)
next
case *Throw* **thus** *?case* **by** *simp*
next
case *Catch* **thus** *?case* **by** (*fastforce intro*: *execn.intros* *elim*!: *execn-elim-cases*)
next
case (*Await* b c e)
have *normalized*: $\Gamma\vdash_p\langle$*normalize* (*Await* b c e),$s\rangle$ =$n\Rightarrow$ t **by** *fact*
hence *exec-norm-a*: $\Gamma\vdash_p\langle$*Await* b (*Language.normalize* c) e,$s\rangle$ =$n\Rightarrow$ t
by *simp*
{
fix s t a
assume *exec-a*: $\Gamma\vdash_p\langle a,s\rangle$ =$n\Rightarrow$ t
have a=*Await* b (*Language.normalize* c) e $\Longrightarrow$ $\Gamma\vdash_p\langle$*Await* b c e,$s\rangle$ =$n\Rightarrow$ t
using *exec-a*
proof (*induct*)
case (*AwaitTrue* s b' $\Gamma 1$ c' n t)
from *AwaitTrue* *execn-normalize-to-execn* **obtain**
s-in-b: $s \in b$ **and**
exec-c: $\Gamma 1\vdash\langle$*Language.normalize* c,*Normal* $s\rangle$ =$n\Rightarrow$ t **and**
hyp-a: $\Gamma\vdash_p\langle$*Await* b c e,*Normal* $s\rangle$ =$n\Rightarrow$ t
using *execn.AwaitTrue* **by** *fastforce*
with *hyp-a* *s-in-b*

have $\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ s\rangle =n\Rightarrow t$
by (*auto intro*: *execn.intros*)
with *AwaitTrue* **show** *?case* **by** *simp*
next
case (*AwaitFalse*) **thus** *?case* **using** *execn.AwaitFalse* **by** *fastforce*
qed (*auto intro*: *execn.intros elim*:*execn-normalize-to-execn*)
}
from *this* [*OF exec-norm-a*]
show *?case*
by *simp*
qed

lemma *execn-normalize-iff-execn*:
$\Gamma\vdash_p\langle normalize\ c,s\rangle =n\Rightarrow t = \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto intro*: *execn-to-execn-normalize execn-normalize-to-execn*)

lemma *exec-sequence-app*:
assumes *exec-xs*: $\Gamma\vdash_p\langle sequence\ Seq\ xs, Normal\ s\rangle \Rightarrow s'$
assumes *exec-ys*: $\Gamma\vdash_p\langle sequence\ Seq\ ys, s'\rangle \Rightarrow t$
shows $\Gamma\vdash_p\langle sequence\ Seq\ (xs@ys), Normal\ s\rangle \Rightarrow t$
proof −
from *exec-to-execn* [*OF exec-xs*]
obtain n **where**
execn-xs: $\Gamma\vdash_p\langle sequence\ Seq\ xs, Normal\ s\rangle =n\Rightarrow s'$..
from *exec-to-execn* [*OF exec-ys*]
obtain m **where**
execn-ys: $\Gamma\vdash_p\langle sequence\ Seq\ ys, s'\rangle =m\Rightarrow t$..
with *execn-xs* **obtain**
$\Gamma\vdash_p\langle sequence\ Seq\ xs, Normal\ s\rangle =max\ n\ m\Rightarrow s'$
$\Gamma\vdash_p\langle sequence\ Seq\ ys, s'\rangle =max\ n\ m\Rightarrow t$
by (*auto intro*: *execn-mono max.cobounded1 max.cobounded2*)
from *execn-sequence-app* [*OF this*]
have $\Gamma\vdash_p\langle sequence\ Seq\ (xs\ @\ ys), Normal\ s\rangle =max\ n\ m\Rightarrow t$.
thus *?thesis*
by (*rule execn-to-exec*)
qed

lemma *exec-sequence-appD*:
assumes *exec-xs-ys*: $\Gamma\vdash_p\langle sequence\ Seq\ (xs\ @\ ys), Normal\ s\rangle \Rightarrow t$
shows $\exists s'.\ \Gamma\vdash_p\langle sequence\ Seq\ xs, Normal\ s\rangle \Rightarrow s' \wedge \Gamma\vdash_p\langle sequence\ Seq\ ys, s'\rangle \Rightarrow t$
proof −
from *exec-to-execn* [*OF exec-xs-ys*]
obtain n **where** $\Gamma\vdash_p\langle sequence\ Seq\ (xs\ @\ ys), Normal\ s\rangle =n\Rightarrow t$..
thus *?thesis*
by (*cases rule*: *execn-sequence-appE*) (*auto intro*: *execn-to-exec*)
qed

lemma *exec-sequence-appE* [*consumes 1*]:
  $[\![\Gamma\vdash_p\langle sequence\ Seq\ (xs\ @\ ys),Normal\ s\rangle \Rightarrow t;$
  $\bigwedge s'.\ [\![\Gamma\vdash_p\langle sequence\ Seq\ xs,Normal\ s\rangle \Rightarrow s';\Gamma\vdash_p\langle sequence\ Seq\ ys,s'\rangle \Rightarrow t]\!] \Longrightarrow P$
  $]\!] \Longrightarrow P$
  by (*auto dest*: *exec-sequence-appD*)

lemma *exec-to-exec-sequence-flatten*:
  assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
  shows $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c),s\rangle \Rightarrow t$
proof −
  from *exec-to-execn* [*OF exec*]
  obtain *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$**..**
  from *execn-to-execn-sequence-flatten* [*OF this*]
  show *?thesis*
    by (*rule execn-to-exec*)
qed

lemma *exec-sequence-flatten-to-exec*:
  assumes *exec-seq*: $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c),s\rangle \Rightarrow t$
  shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
proof −
  from *exec-to-execn* [*OF exec-seq*]
  obtain *n* **where** $\Gamma\vdash_p\langle sequence\ Seq\ (flatten\ c),s\rangle =n\Rightarrow t$**..**
  from *execn-sequence-flatten-to-execn* [*OF this*]
  show *?thesis*
    by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-normalize*:
  assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
  shows $\Gamma\vdash_p\langle normalize\ c,s\rangle \Rightarrow t$
proof −
  from *exec-to-execn* [*OF exec*] **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$**..**
  hence $\Gamma\vdash_p\langle normalize\ c,s\rangle =n\Rightarrow t$
    by (*rule execn-to-execn-normalize*)
  thus *?thesis*
    by (*rule execn-to-exec*)
qed

lemma *exec-normalize-to-exec*:
  assumes *exec*: $\Gamma\vdash_p\langle normalize\ c,s\rangle \Rightarrow t$
  shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
proof −
  from *exec-to-execn* [*OF exec*] **obtain** *n* **where** $\Gamma\vdash_p\langle normalize\ c,s\rangle =n\Rightarrow t$**..**
  hence $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
    by (*rule execn-normalize-to-execn*)
  thus *?thesis*
    by (*rule execn-to-exec*)

qed

lemma *exec-normalize-iff-exec*:
$\Gamma\vdash_p\langle normalize\ c,s\rangle \Rightarrow t = \Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
by (*auto intro*: *exec-to-exec-normalize exec-normalize-to-exec*)

6.4 Lemmas about $c_1 \subseteq_g c_2$

lemma *execn-to-execn-subseteq-guards*: $\bigwedge c\ s\ t\ n.\ [\![c \subseteq_{gs} c';\ \Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t]\!]$
$\Longrightarrow \exists\, t'.\ \Gamma\vdash_p\langle c',s\rangle =n\Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge (\neg\ isFault\ t' \longrightarrow t'=t)$
proof (*induct c'*)
case *Skip* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case *Basic* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case *Spec* **thus** *?case*
by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
case (*Seq c1' c2'*)
have $c \subseteq_{gs}$ *Seq c1' c2'* **by** *fact*
from *subseteq-guards-Seq* [*OF this*]
obtain *c1 c2* **where**
c: *c = Seq c1 c2* **and**
c1-c1': *c1* $\subseteq_{gs}$ *c1'* **and**
c2-c2': *c2* $\subseteq_{gs}$ *c2'*
by *blast*
have *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **obtain** *w* **where**
exec-c1: $\Gamma\vdash_p\langle c1,s\rangle =n\Rightarrow w$ **and**
exec-c2: $\Gamma\vdash_p\langle c2,w\rangle =n\Rightarrow t$
by (*auto elim*: *execn-elim-cases*)
from *exec-c1 Seq.hyps c1-c1'*
obtain *w'* **where**
exec-c1': $\Gamma\vdash_p\langle c1',s\rangle =n\Rightarrow w'$ **and**
w-Fault: *isFault w* $\longrightarrow$ *isFault w'* **and**
w'-noFault: $\neg$ *isFault w'* $\longrightarrow$ *w'=w*
by *blast*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*

```
  with exec have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  show ?thesis
  proof (cases isFault w)
    case True
    then obtain f where w': w=Fault f..
    moreover with exec-c2
    have t: t=Fault f
      by (auto dest: execn-Fault-end)
    ultimately show ?thesis
      using Normal w-Fault exec-c1'
      by (fastforce intro: execn.intros elim: isFaultE)
  next
    case False
    note noFault-w = this
    show ?thesis
    proof (cases isFault w')
      case True
      then obtain f' where w': w'=Fault f'..
      with Normal exec-c1'
      have exec: Γ⊢p⟨Seq c1' c2',s⟩ =n⇒ Fault f'
        by (auto intro: execn.intros)
      then show ?thesis
        by auto
    next
      case False
      with w'-noFault have w': w'=w by simp
      from Seq.hyps exec-c2 c2-c2'
      obtain t' where
        Γ⊢p⟨c2',w⟩ =n⇒ t' and
        isFault t ⟶ isFault t' and
        ¬ isFault t' ⟶ t'=t
        by blast
      with Normal exec-c1' w'
      show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
qed
```

next
  case (*Cond b c1' c2'*)
  have *exec*: $\Gamma\vdash_p\langle c,s\rangle$ $=n\Rightarrow$ t **by** *fact*
  have $c \subseteq_{gs}$ *Cond b c1' c2'* **by** *fact*
  from *subseteq-guards-Cond* [*OF this*]
  obtain *c1 c2* **where**
    c: c = *Cond b c1 c2* **and**
    c1-c1': *c1* $\subseteq_{gs}$ *c1'* **and**
    c2-c2': *c2* $\subseteq_{gs}$ *c2'*
    by *blast*
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec* **have** *t=Fault f*
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec* **have** *t=Stuck*
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case (*Abrupt s'*)
    with *exec* **have** *t=Abrupt s'*
      by (*auto dest*: *execn-Abrupt-end*)
    with *Abrupt* **show** *?thesis*
      by *auto*
  next
    case (*Normal s'*)
    from *exec* [*simplified c Normal*]
    show *?thesis*
    proof (*cases*)
      assume *s'-in-b*: $s' \in b$
      assume $\Gamma\vdash_p\langle c1,Normal\ s'\rangle$ $=n\Rightarrow$ t
      with *c1-c1' Normal Cond.hyps* **obtain** t' **where**
        $\Gamma\vdash_p\langle c1',Normal\ s'\rangle$ $=n\Rightarrow$ t'
        isFault t $\longrightarrow$ *isFault t'*
        $\neg$ *isFault t'* $\longrightarrow$ $t' = t$
        by *blast*
      with *s'-in-b Normal* **show** *?thesis*
        by (*fastforce intro*: *execn.intros*)
    next
      assume *s'-notin-b*: $s' \notin b$
      assume $\Gamma\vdash_p\langle c2,Normal\ s'\rangle$ $=n\Rightarrow$ t
      with *c2-c2' Normal Cond.hyps* **obtain** t' **where**
        $\Gamma\vdash_p\langle c2',Normal\ s'\rangle$ $=n\Rightarrow$ t'
        isFault t $\longrightarrow$ *isFault t'*

$\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
with *s′-notin-b Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
next
case (*While b c′*)
have *exec*: $\Gamma\vdash_p\langle c,s\rangle$ $=n\Rightarrow$ t **by** *fact*
have $c \subseteq_{gs}$ *While b c′* **by** *fact*
from *subseteq-guards-While* [*OF this*]
obtain c'' **where**
c: c = *While b c″* **and**
c″-c′: $c'' \subseteq_{gs} c'$
by *blast*
{
fix *c r w*
assume *exec*: $\Gamma\vdash_p\langle c,r\rangle$ $=n\Rightarrow$ w
assume *c*: c=*While b c″*
have $\exists w'$. $\Gamma\vdash_p\langle$*While b c′*,$r\rangle$ $=n\Rightarrow$ $w' \wedge$
(*isFault* $w \longrightarrow$ *isFault* w') $\wedge$ ($\neg$ *isFault* $w' \longrightarrow w'=w$)
using *exec c*
proof (*induct*)
case (*WhileTrue r b′ ca n u w*)
have *eqs*: *While b′ ca* = *While b c″* **by** *fact*
from *WhileTrue* **have** *r-in-b*: $r \in b$ **by** *simp*
from *WhileTrue* **have** *exec-c″*: $\Gamma\vdash_p\langle c''$,*Normal r*$\rangle$ $=n\Rightarrow$ u **by** *simp*
from *While.hyps* [*OF c″-c′ exec-c″*] **obtain** u' **where**
exec-c′: $\Gamma\vdash_p\langle c'$,*Normal r*$\rangle$ $=n\Rightarrow$ u' **and**
u-Fault: *isFault* $u \longrightarrow$ *isFault* u' **and**
u′-noFault: $\neg$ *isFault* $u' \longrightarrow u' = u$
by *blast*
from *WhileTrue* **obtain** w' **where**
exec-w: $\Gamma\vdash_p\langle$*While b c′*,$u\rangle$ $=n\Rightarrow$ w' **and**
w-Fault: *isFault* $w \longrightarrow$ *isFault* w' **and**
w′-noFault: $\neg$ *isFault* $w' \longrightarrow w' = w$
by *blast*
show *?case*
proof (*cases isFault u′*)
case *True*
with *exec-c′ r-in-b*
show *?thesis*
by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
case *False*
with *exec-c′ r-in-b u′-noFault exec-w w-Fault w′-noFault*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed

    next
      case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
    qed *auto*
  }
  from *this* [*OF exec c*]
  show *?case* **.**
next
  case *Call* **thus** *?case*
    by (*fastforce dest*: *subseteq-guardsD elim*: *execn-elim-cases*)
next
  case (*DynCom C'*)
  have *exec*: $\Gamma\vdash_p\langle c,s\rangle$ $=n\Rightarrow$ t **by** *fact*
  have $c \subseteq_{gs}$ *DynCom C'* **by** *fact*
  from *subseteq-guards-DynCom* [*OF this*] **obtain** *C* **where**
    c: *c* = *DynCom C* **and**
    C-C': $\forall s.\ C\ s \subseteq_{gs} C'\ s$
    by *blast*
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec* **have** *t=Fault f*
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec* **have** *t=Stuck*
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case (*Abrupt s'*)
    with *exec* **have** *t=Abrupt s'*
      by (*auto dest*: *execn-Abrupt-end*)
    with *Abrupt* **show** *?thesis*
      by *auto*
  next
    case (*Normal s'*)
    from *exec* [*simplified c Normal*]
    have $\Gamma\vdash_p\langle C\ s',Normal\ s'\rangle$ $=n\Rightarrow$ t
      by *cases*
    from *DynCom.hyps C-C'* [*rule-format*] *this* **obtain** *t'* **where**
      $\Gamma\vdash_p\langle C'\ s',Normal\ s'\rangle$ $=n\Rightarrow$ t'
      isFault t $\longrightarrow$ *isFault t'*
      $\neg$ *isFault t'* $\longrightarrow$ $t' = t$
      by *blast*
    with *Normal* **show** *?thesis*
      by (*fastforce intro*: *execn.intros*)
  qed

next
  case $(Guard\ f'\ g'\ c')$
  have *exec*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **by** *fact*
  have $c \subseteq_{gs} Guard\ f'\ g'\ c'$ **by** *fact*
  hence *subset-cases*: $(c \subseteq_{gs} c') \lor (\exists\, c''.\ c = Guard\ f'\ g'\ c'' \land (c'' \subseteq_{gs} c'))$
    by (*rule subseteq-guards-Guard*)
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec* **have** $t{=}Fault\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec* **have** $t{=}Stuck$
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case $(Abrupt\ s')$
    with *exec* **have** $t{=}Abrupt\ s'$
      by (*auto dest*: *execn-Abrupt-end*)
    with *Abrupt* **show** *?thesis*
      by *auto*
  next
    case $(Normal\ s')$
    from *subset-cases* **show** *?thesis*
    proof
      assume *c-c'*: $c \subseteq_{gs} c'$
      from *Guard.hyps* [*OF this exec*] *Normal* **obtain** t' **where**
        exec-c': $\Gamma\vdash_p\langle c',Normal\ s'\rangle =n\Rightarrow t'$ **and**
        t-Fault: $isFault\ t \longrightarrow isFault\ t'$ **and**
        t-noFault: $\neg\ isFault\ t' \longrightarrow t' = t$
        by *blast*
      with *Normal*
      show *?thesis*
        by $(cases\ s' \in g')$ (*fastforce intro*: *execn.intros*)+
    next
      assume $\exists\, c''.\ c = Guard\ f'\ g'\ c'' \land (c'' \subseteq_{gs} c')$
      then obtain c'' **where**
        c: $c = Guard\ f'\ g'\ c''$ **and**
        c''-c': $c'' \subseteq_{gs} c'$
        by *blast*
      from *c exec Normal*
      have *exec-Guard'*: $\Gamma\vdash_p\langle Guard\ f'\ g'\ c'',Normal\ s'\rangle =n\Rightarrow t$
        by *simp*
      thus *?thesis*
      proof (*cases*)

```
      assume s'-in-g': s' ∈ g'
      assume exec-c'': Γ⊢p⟨c'',Normal s'⟩ =n⇒ t
      from Guard.hyps [OF c''-c' exec-c''] obtain t' where
        exec-c': Γ⊢p⟨c',Normal s'⟩ =n⇒ t' and
        t-Fault: isFault t ⟶ isFault t' and
        t-noFault: ¬ isFault t' ⟶ t' = t
        by blast
      with Normal s'-in-g'
      show ?thesis
        by (fastforce intro: execn.intros)
    next
      assume s' ∉ g' t=Fault f'
      with Normal show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
 qed
next
 case Throw thus ?case
  by (fastforce dest: subseteq-guardsD intro: execn.intros
      elim: execn-elim-cases)
next
 case (Catch c1' c2')
 have c ⊆gs Catch c1' c2' by fact
 from subseteq-guards-Catch [OF this]
 obtain c1 c2 where
  c: c = Catch c1 c2 and
  c1-c1': c1 ⊆gs c1' and
  c2-c2': c2 ⊆gs c2'
  by blast
 have exec: Γ⊢p⟨c,s⟩ =n⇒ t by fact
 show ?case
 proof (cases s)
  case (Fault f)
  with exec have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
 next
  case Stuck
  with exec have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
 next
  case (Abrupt s')
  with exec have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
```

by *auto*
next
case (*Normal s'*)
from *exec* [*simplified c Normal*]
show *?thesis*
proof (*cases*)
fix *w*
assume *exec-c1*: $\Gamma\vdash_p\langle c1, Normal\ s'\rangle =n\Rightarrow Abrupt\ w$
assume *exec-c2*: $\Gamma\vdash_p\langle c2, Normal\ w\rangle =n\Rightarrow t$
from *Normal exec-c1 c1-c1' Catch.hyps* **obtain** *w'* **where**
exec-c1': $\Gamma\vdash_p\langle c1', Normal\ s'\rangle =n\Rightarrow w'$ **and**
w'-noFault: $\neg\ isFault\ w' \longrightarrow w' = Abrupt\ w$
by *blast*
show *?thesis*
proof (*cases isFault w'*)
case *True*
with *exec-c1' Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
case *False*
with *w'-noFault* **have** *w'*: $w'=Abrupt\ w$ **by** *simp*
from *Normal exec-c2 c2-c2' Catch.hyps* **obtain** *t'* **where**
$\Gamma\vdash_p\langle c2', Normal\ w\rangle =n\Rightarrow t'$
$isFault\ t \longrightarrow isFault\ t'$
$\neg\ isFault\ t' \longrightarrow t' = t$
by *blast*
with *exec-c1' w' Normal*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
next
assume *exec-c1*: $\Gamma\vdash_p\langle c1, Normal\ s'\rangle =n\Rightarrow t$
assume *t*: $\neg\ isAbr\ t$
from *Normal exec-c1 c1-c1' Catch.hyps* **obtain** *t'* **where**
exec-c1': $\Gamma\vdash_p\langle c1', Normal\ s'\rangle =n\Rightarrow t'$ **and**
t-Fault: $isFault\ t \longrightarrow isFault\ t'$ **and**
t'-noFault: $\neg\ isFault\ t' \longrightarrow t' = t$
by *blast*
show *?thesis*
proof (*cases isFault t'*)
case *True*
with *exec-c1' Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
case *False*
with *exec-c1' Normal t-Fault t'-noFault t*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed

qed
qed
next
case (*Await b c' e*)
then obtain c'' **where** *c-Await*:c=*Await b* c'' *e* $\wedge$ ($c'' \subseteq_g c'$) **using** *subseteq-guards-Await* **by** *blast*
thus *?case*
proof (*cases s*)
case *Abrupt* **thus** *?thesis*
using *Await.prems*(*2*) *SemanticCon.execn-Abrupt-end* **by** *fastforce*
next
case *Stuck* **thus** *?thesis*
using *Await.prems*(*2*) *SemanticCon.execn-Stuck-end* **by** *blast*
next
case *Fault* **thus** *?thesis* **by** *auto*
next
case (*Normal x*) **thus** *?thesis*
proof (*cases* $x \in b$)
case *True*
then obtain $\Gamma 1$ **where** $\Gamma 1 \vdash \langle c'',s\rangle =n\Rightarrow t$ **using** *c-Await Await*
by (*metis Normal SemanticCon.execn-Normal-elim-cases*(*11*))
then obtain t' **where** $\Gamma 1 \vdash \langle c',s\rangle =n\Rightarrow t' \wedge$
(*Semantic.isFault t* $\longrightarrow$ *Semantic.isFault* t') $\wedge$ ($\neg$ *Semantic.isFault* t' $\longrightarrow t' = t$)
using *Semantic.execn-to-execn-subseteq-guards c-Await* **by** *blast*
thus *?thesis* **using** *Await.prems*(*1*) *Await.prems*(*2*) *c-Await True SemanticCon.execn-Normal-elim-cases*(*11*)
by (*metis Normal Semantic.isFaultE SemanticCon.isFault-simps*(*3*) *execn.AwaitTrue execn-to-execn-subseteq-guards*)
next
case *False*
then show $\exists t'. \Gamma\vdash_p \langle Await\ b\ c'\ e,s\rangle =n\Rightarrow t' \wedge$
(*SemanticCon.isFault t* $\longrightarrow$ *SemanticCon.isFault* t') $\wedge$
($\neg$ *SemanticCon.isFault* $t' \longrightarrow t' = t$) **using** *False execn-Normal-elim-cases*(*11*)
by (*metis Await.prems*(*2*) *Normal c-Await execn.AwaitFalse*)
qed
qed
qed

lemma *exec-to-exec-subseteq-guards*:
assumes *c-c'*: $c \subseteq_{gs} c'$
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
shows $\exists t'. \Gamma\vdash_p\langle c',s\rangle \Rightarrow t' \wedge$
(*isFault t* $\longrightarrow$ *isFault* t') $\wedge$ ($\neg$ *isFault* $t' \longrightarrow t'=t$)
proof –
from *exec-to-execn* [*OF exec*] **obtain** *n* **where**
$\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **..**
from *execn-to-execn-subseteq-guards* [*OF c-c' this*]

show *?thesis*
  by (*blast intro*: *execn-to-exec*)
qed

6.5 Lemmas about *LanguageCon.merge-guards*

theorem *execn-to-execn-merge-guards*:
 assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
 shows $\Gamma\vdash_p\langle \mathit{merge\text{-}guards}\ c,s\rangle =n\Rightarrow t$
using *exec-c*
proof (*induct*)
  case (*Guard s g c n t f*)
  have *s-in-g*: $s \in g$ **by** *fact*
  have *exec-merge-c*: $\Gamma\vdash_p\langle \mathit{merge\text{-}guards}\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases* $\exists f'\ g'\ c'.\ \mathit{merge\text{-}guards}\ c = \mathit{Guard}\ f'\ g'\ c'$)
    case *False*
    with *exec-merge-c s-in-g*
    show *?thesis*
      by (*cases merge-guards c*) (*auto intro*: *execn.intros simp add*: *Let-def*)
  next
    case *True*
    then obtain $f'\ g'\ c'$ **where**
      merge-guards-c: $\mathit{merge\text{-}guards}\ c = \mathit{Guard}\ f'\ g'\ c'$
      by *iprover*
    show *?thesis*
    proof (*cases* $f=f'$)
      case *False*
      from *exec-merge-c s-in-g merge-guards-c False* **show** *?thesis*
        by (*auto intro*: *execn.intros simp add*: *Let-def*)
    next
      case *True*
      from *exec-merge-c s-in-g merge-guards-c True* **show** *?thesis*
        by (*fastforce intro*: *execn.intros elim*: *execn.cases*)
    qed
  qed
next
  case (*GuardFault s g f c n*)
  have *s-notin-g*: $s \notin g$ **by** *fact*
  show *?case*
  proof (*cases* $\exists f'\ g'\ c'.\ \mathit{merge\text{-}guards}\ c = \mathit{Guard}\ f'\ g'\ c'$)
    case *False*
    with *s-notin-g*
    show *?thesis*
      by (*cases merge-guards c*) (*auto intro*: *execn.intros simp add*: *Let-def*)
  next
    case *True*
    then obtain $f'\ g'\ c'$ **where**
      merge-guards-c: $\mathit{merge\text{-}guards}\ c = \mathit{Guard}\ f'\ g'\ c'$

    by *iprover*
  show *?thesis*
  proof (*cases* $f=f'$)
    case *False*
    from *s-notin-g merge-guards-c False* **show** *?thesis*
      by (*auto intro*: *execn.intros simp add*: *Let-def*)
  next
    case *True*
    from *s-notin-g merge-guards-c True* **show** *?thesis*
      by (*fastforce intro*: *execn.intros*)
  qed
 qed
next
 case (*AwaitTrue s b* $\Gamma 1$ *c n t*)
 then have $\Gamma 1\vdash$ $\langle$*Language.merge-guards c*,*Normal s*$\rangle$ $=n\Rightarrow$ *t*
   by (*simp add*: *AwaitTrue.hyps*(*2*) *execn-to-execn-merge-guards*)
 thus *?case*
   by (*simp add*: *AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *execn.AwaitTrue*)
qed (*fastforce intro*: *execn.intros*)+

lemma *execn-merge-guards-to-execn-Normal*:
 $\bigwedge s\ n\ t.\ \Gamma\vdash_p\langle merge\text{-}guards\ c,Normal\ s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$
proof (*induct c*)
 case *Skip* **thus** *?case* **by** *auto*
next
 case *Basic* **thus** *?case* **by** *auto*
next
 case *Spec* **thus** *?case* **by** *auto*
next
 case (*Seq c1 c2*)
 have $\Gamma\vdash_p\langle$*merge-guards* (*Seq c1 c2*),*Normal s*$\rangle$ $=n\Rightarrow t$ **by** *fact*
 hence *exec-merge*: $\Gamma\vdash_p\langle$*Seq* (*merge-guards c1*) (*merge-guards c2*),*Normal s*$\rangle$
$=n\Rightarrow t$
   by *simp*
 then obtain s' **where**
   exec-merge-c1: $\Gamma\vdash_p\langle$*merge-guards c1*,*Normal s*$\rangle$ $=n\Rightarrow s'$ **and**
   exec-merge-c2: $\Gamma\vdash_p\langle$*merge-guards c2*,$s'\rangle$ $=n\Rightarrow t$
   by *cases*
 from *exec-merge-c1*
 have *exec-c1*: $\Gamma\vdash_p\langle c1$,*Normal s*$\rangle$ $=n\Rightarrow s'$
   by (*rule Seq.hyps*)
 show *?case*
 proof (*cases* s')
   case (*Normal* s'')
   with *exec-merge-c2*
   have $\Gamma\vdash_p\langle c2,s'\rangle =n\Rightarrow t$
     by (*auto intro*: *Seq.hyps*)
   with *exec-c1* **show** *?thesis*

```
      by (auto intro: execn.intros)
  next
    case (Abrupt s'')
    with exec-merge-c2 have t=Abrupt s''
      by (auto dest: execn-Abrupt-end)
    with exec-c1 Abrupt
    show ?thesis
      by (auto intro: execn.intros)
  next
    case (Fault f)
    with exec-merge-c2 have t=Fault f
      by (auto dest: execn-Fault-end)
    with exec-c1 Fault
    show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-merge-c2 have t=Stuck
      by (auto dest: execn-Stuck-end)
    with exec-c1 Stuck
    show ?thesis
      by (auto intro: execn.intros)
  qed
next
  case Cond thus ?case
    by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
  case (While b c)
  {
    fix c' r w
    assume exec-c': Γ⊢p⟨c',r⟩ =n⇒ w
    assume c': c'=While b (merge-guards c)
    have Γ⊢p⟨While b c,r⟩ =n⇒ w
      using exec-c' c'
    proof (induct)
      case (WhileTrue r b' c'' n u w)
      have eqs: While b' c'' = While b (merge-guards c) by fact
      from WhileTrue
      have r-in-b: r ∈ b
        by simp
      from WhileTrue While.hyps have exec-c: Γ⊢p⟨c,Normal r⟩ =n⇒ u
        by simp
      from WhileTrue have exec-w: Γ⊢p⟨While b c,u⟩ =n⇒ w
        by simp
      from r-in-b exec-c exec-w
      show ?case
        by (rule execn.WhileTrue)
    next
      case WhileFalse thus ?case by (auto intro: execn.WhileFalse)
```

```
    qed auto
  }
  with While.prems show ?case
    by (auto)
next
  case Call thus ?case by simp
next
  case DynCom thus ?case
    by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
  case (Guard f g c)
  have exec-merge: Γ⊢p⟨merge-guards (Guard f g c),Normal s⟩ =n⇒ t by fact
  show ?case
  proof (cases s ∈ g)
    case False
    with exec-merge have t=Fault f
      by (auto split: com.splits if-split-asm elim: execn-Normal-elim-cases
        simp add: Let-def is-Guard-def)
    with False show ?thesis
      by (auto intro: execn.intros)
  next
    case True
    note s-in-g = this
    show ?thesis
    proof (cases ∃f' g' c'. merge-guards c = Guard f' g' c')
      case False
      then
      have merge-guards (Guard f g c) = Guard f g (merge-guards c)
        by (cases merge-guards c) (auto simp add: Let-def)
      with exec-merge s-in-g
      obtain Γ⊢p⟨merge-guards c,Normal s⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      from Guard.hyps [OF this] s-in-g
      show ?thesis
        by (auto intro: execn.intros)
    next
      case True
      then obtain f' g' c' where
        merge-guards-c: merge-guards c = Guard f' g' c'
        by iprover
      show ?thesis
      proof (cases f=f')
        case False
        with merge-guards-c
        have merge-guards (Guard f g c) = Guard f g (merge-guards c)
          by (simp add: Let-def)
        with exec-merge s-in-g
        obtain Γ⊢p⟨merge-guards c,Normal s⟩ =n⇒ t
          by (auto elim: execn-Normal-elim-cases)
```

```
        from Guard.hyps [OF this] s-in-g
        show ?thesis
          by (auto intro: execn.intros)
      next
        case True
        note f-eq-f' = this
        with merge-guards-c have
          merge-guards-Guard: merge-guards (Guard f g c) = Guard f (g ∩ g') c'
          by simp
        show ?thesis
        proof (cases s ∈ g')
          case True
          with exec-merge merge-guards-Guard merge-guards-c s-in-g
          have Γ⊢p⟨merge-guards c,Normal s⟩ =n⇒ t
            by (auto intro: execn.intros elim: execn-Normal-elim-cases)
          with Guard.hyps [OF this] s-in-g
          show ?thesis
            by (auto intro: execn.intros)
        next
          case False
          with exec-merge merge-guards-Guard
          have t=Fault f
            by (auto elim: execn-Normal-elim-cases)
          with merge-guards-c f-eq-f' False
          have Γ⊢p⟨merge-guards c,Normal s⟩ =n⇒ t
            by (auto intro: execn.intros)
          from Guard.hyps [OF this] s-in-g
          show ?thesis
            by (auto intro: execn.intros)
        qed
      qed
    qed
  qed
next
  case Throw thus ?case by simp
next
  case (Catch c1 c2)
  have Γ⊢p⟨merge-guards (Catch c1 c2),Normal s⟩ =n⇒ t  by fact
  hence Γ⊢p⟨Catch (merge-guards c1) (merge-guards c2),Normal s⟩ =n⇒ t by
simp
  thus ?case
    by cases (auto intro: execn.intros Catch.hyps)
next
  case (Await b c e)
  {
    fix c' r w
    assume exec-c': Γ⊢p⟨c',r⟩ =n⇒ w
    assume c': c'=Await b (Language.merge-guards c) e
    have Γ⊢p⟨Await b c e,r⟩ =n⇒ w
```

using *exec-c′ c′*
proof (*induct*)
case (*AwaitTrue r b′ Γ1 c′′ n u*)
then have *eqs*: *Await b′ c′′ e* = *Await b* (*Language.merge-guards c*) *e* **by** *auto*
from *AwaitTrue*
have *r-in-b*: $r \in b$
by *simp*
from *AwaitTrue* **have** *exec-c*: $\Gamma 1 \vdash \langle c, Normal\ r\rangle =n\Rightarrow u$
using *execn-merge-guards-to-execn* **by** *force*
then have $\Gamma_{\neg a}\vdash \langle c, Normal\ r\rangle =n\Rightarrow u$ **using** *AwaitTrue.hyps*(*2*) *exec-c* **by** *blast*
then have *exec-a*: $\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ r\rangle =n\Rightarrow u$
by (*meson exec-c execn.AwaitTrue r-in-b*)
from *r-in-b exec-c exec-a*
show *?case*
by (*simp add*: *execn.AwaitTrue*)
next
case (*AwaitFalse b c*) **thus** *?case* **by** (*simp add*: *execn.AwaitFalse*)
qed *auto*
}
with *Await.prems* **show** *?case*
by (*auto*)
qed

theorem *execn-merge-guards-to-execn*:
$\Gamma\vdash_p\langle merge\text{-}guards\ c,s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash_p\langle c,\ s\rangle =n\Rightarrow t$
apply (*cases s*)
apply (*fastforce intro*: *execn-merge-guards-to-execn-Normal*)
apply (*fastforce dest*: *execn-Abrupt-end*)
apply (*fastforce dest*: *execn-Fault-end*)
apply (*fastforce dest*: *execn-Stuck-end*)
done

corollary *execn-iff-execn-merge-guards*:
$\Gamma\vdash_p\langle c,\ s\rangle =n\Rightarrow t = \Gamma\vdash_p\langle merge\text{-}guards\ c,s\rangle =n\Rightarrow t$
by (*blast intro*: *execn-merge-guards-to-execn execn-to-execn-merge-guards*)

theorem *exec-iff-exec-merge-guards*:
$\Gamma\vdash_p\langle c,\ s\rangle \Rightarrow t = \Gamma\vdash_p\langle merge\text{-}guards\ c,s\rangle \Rightarrow t$
by (*blast dest*: *exec-to-execn intro*: *execn-to-exec*
intro: *execn-to-execn-merge-guards*
execn-merge-guards-to-execn)

corollary *exec-to-exec-merge-guards*:
$\Gamma\vdash_p\langle c,\ s\rangle \Rightarrow t \Longrightarrow \Gamma\vdash_p\langle merge\text{-}guards\ c,s\rangle \Rightarrow t$
by (*rule iffD1* [*OF exec-iff-exec-merge-guards*])

corollary *exec-merge-guards-to-exec*:

$\Gamma\vdash_p\langle$*merge-guards c*,*s*$\rangle \Rightarrow t \Longrightarrow \Gamma\vdash_p\langle c,\ s\rangle \Rightarrow t$
 by (*rule iffD2* [*OF exec-iff-exec-merge-guards*])

6.6 Lemmas about *LanguageCon.mark-guards*

lemma *execn-to-execn-mark-guards*:
 assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
 assumes *t-not-Fault*: $\neg$ *isFault t*
 shows $\Gamma\vdash_p\langle$*mark-guards f c*,*s*$\rangle =n\Rightarrow t$
using *exec-c t-not-Fault* [*simplified not-isFault-iff*]
proof *induct*
 case (*AwaitTrue s b* Γ*1 c n t*)
 then have Γ*1*$\vdash$ $\langle$*Language.mark-guards f c*,*Normal s*$\rangle =n\Rightarrow t$
   by (*meson Semantic.isFaultE execn-to-execn-mark-guards*)
 thus *?case* **by** (*auto intro*:*AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *execn.AwaitTrue*)
qed(*auto intro*: *execn.intros dest*: *noFaultn-startD'*)

lemma *execn-to-execn-mark-guards-Fault*:
 assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
 shows $\bigwedge f.\ [\![t=Fault\ f]\!] \Longrightarrow \exists f'.\ \Gamma\vdash_p\langle$*mark-guards x c*,*s*$\rangle =n\Rightarrow$ *Fault f'*
using *exec-c*
proof (*induct*)
  case *Skip* **thus** *?case* **by** *auto*
next
  case *Guard* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *FaultProp* **thus** *?case* **by** *auto*
next
 case *Basic* **thus** *?case* **by** *auto*
next
 case *Spec* **thus** *?case* **by** *auto*
next
 case *SpecStuck* **thus** *?case* **by** *auto*
next
  case (*Seq c1 s n w c2 t*)
  have *exec-c1*: $\Gamma\vdash_p\langle c1,$*Normal s*$\rangle =n\Rightarrow w$ **by** *fact*
  have *exec-c2*: $\Gamma\vdash_p\langle c2,w\rangle =n\Rightarrow t$ **by** *fact*
  have *t*: *t*=*Fault f* **by** *fact*
  show *?case*
  proof (*cases w*)
    case (*Fault f'*)
    with *exec-c2 t* **have** *f'*=*f*
      by (*auto dest*: *execn-Fault-end*)
    with *Fault Seq.hyps* **obtain** *f''* **where**
      $\Gamma\vdash_p\langle$*mark-guards x c1*,*Normal s*$\rangle =n\Rightarrow$ *Fault f''*
      by *auto*
    moreover have $\Gamma\vdash_p\langle$*mark-guards x c2*,*Fault f''*$\rangle =n\Rightarrow$ *Fault f''*

by *auto*
ultimately show *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Normal s'*)
with *execn-to-execn-mark-guards* [*OF exec-c1*]
have *exec-mark-c1*: $\Gamma\vdash_p\langle mark\text{-}guards\ x\ c1,Normal\ s\rangle =n\Rightarrow w$
by *simp*
with *Seq.hyps t* **obtain** f' **where**
$\Gamma\vdash_p\langle mark\text{-}guards\ x\ c2,w\rangle =n\Rightarrow Fault\ f'$
by *blast*
with *exec-mark-c1* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Abrupt s'*)
with *execn-to-execn-mark-guards* [*OF exec-c1*]
have *exec-mark-c1*: $\Gamma\vdash_p\langle mark\text{-}guards\ x\ c1,Normal\ s\rangle =n\Rightarrow w$
by *simp*
with *Seq.hyps t* **obtain** f' **where**
$\Gamma\vdash_p\langle mark\text{-}guards\ x\ c2,w\rangle =n\Rightarrow Fault\ f'$
by (*auto intro*: *execn.intros*)
with *exec-mark-c1* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-c2* **have** $t{=}Stuck$
by (*auto dest*: *execn-Stuck-end*)
with *t* **show** *?thesis* **by** *simp*
qed
next
case *CondTrue* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *CondFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case (*WhileTrue s b c n w t*)
have *exec-c*: $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow w$ **by** *fact*
have *exec-w*: $\Gamma\vdash_p\langle While\ b\ c,w\rangle =n\Rightarrow t$ **by** *fact*
have *t*: $t = Fault\ f$ **by** *fact*
have *s-in-b*: $s \in b$ **by** *fact*
show *?case*
proof (*cases w*)
case (*Fault f'*)
with *exec-w t* **have** $f'{=}f$
by (*auto dest*: *execn-Fault-end*)
with *Fault WhileTrue.hyps* **obtain** f'' **where**
$\Gamma\vdash_p\langle mark\text{-}guards\ x\ c,Normal\ s\rangle =n\Rightarrow Fault\ f''$
by *auto*
moreover have $\Gamma\vdash_p\langle mark\text{-}guards\ x\ (While\ b\ c),Fault\ f''\rangle =n\Rightarrow Fault\ f''$
by *auto*

```
    ultimately show ?thesis
      using s-in-b by (auto intro: execn.intros)
  next
    case (Normal s')
    with execn-to-execn-mark-guards [OF exec-c]
    have exec-mark-c: Γ⊢p⟨mark-guards x c,Normal s⟩ =n⇒ w
      by simp
    with WhileTrue.hyps t obtain f' where
      Γ⊢p⟨mark-guards x (While b c),w⟩ =n⇒ Fault f'
      by blast
    with exec-mark-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-mark-guards [OF exec-c]
    have exec-mark-c: Γ⊢p⟨mark-guards x c,Normal s⟩ =n⇒ w
      by simp
    with WhileTrue.hyps t obtain f' where
      Γ⊢p⟨mark-guards x (While b c),w⟩ =n⇒ Fault f'
      by (auto intro: execn.intros)
    with exec-mark-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-w have t=Stuck
      by (auto dest: execn-Stuck-end)
    with t show ?thesis by simp
  qed
next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
next
  case Call thus ?case by (fastforce intro: execn.intros)
next
  case CallUndefined thus ?case by simp
next
  case StuckProp thus ?case by simp
next
  case DynCom thus ?case by (fastforce intro: execn.intros)
next
  case Throw thus ?case by simp
next
  case AbruptProp thus ?case by simp
next
  case (CatchMatch c1 s n w c2 t)
  have exec-c1: Γ⊢p⟨c1,Normal s⟩ =n⇒ Abrupt w by fact
  have exec-c2: Γ⊢p⟨c2,Normal w⟩ =n⇒ t by fact
  have t: t = Fault f by fact
  from execn-to-execn-mark-guards [OF exec-c1]
  have exec-mark-c1: Γ⊢p⟨mark-guards x c1,Normal s⟩ =n⇒ Abrupt w
```

by *simp*
with *CatchMatch.hyps t* **obtain** f' **where**
$\Gamma\vdash_p\langle$*mark-guards x c2*,*Normal w*$\rangle$ =*n*$\Rightarrow$ *Fault* f'
by *blast*
with *exec-mark-c1* **show** *?case*
by (*auto intro*: *execn.intros*)
next
case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case (*AwaitTrue s b* Γ*1 c n t*)
then have $\exists f'$. Γ*1*$\vdash\langle$*Language.mark-guards x c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *Fault* f'
by (*simp add*: *execn-to-execn-mark-guards-Fault*)
thus *?case* **using** *AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *execn.AwaitTrue* **by** *fastforce*
next
case (*AwaitFalse s b*) **thus** *?case* **by** (*auto simp add*:*execn.AwaitFalse*)
qed

lemma *execn-mark-guards-to-execn*:
$\bigwedge s\ n\ t$. $\Gamma\vdash_p\langle$*mark-guards f c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
$\Longrightarrow \exists\, t'$. $\Gamma\vdash_p\langle c,s\rangle$ =*n*$\Rightarrow$ $t' \wedge$
(*isFault t* $\longrightarrow$ *isFault* t') $\wedge$
(t' = *Fault f* $\longrightarrow$ t'=*t*) $\wedge$
(*isFault* t' $\longrightarrow$ *isFault t*) $\wedge$
($\neg$ *isFault* t' $\longrightarrow$ t'=*t*)
proof (*induct c*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case (*Seq c1 c2 s n t*)
have *exec-mark*: $\Gamma\vdash_p\langle$*mark-guards f* (*Seq c1 c2*),*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
then obtain *w* **where**
exec-mark-c1: $\Gamma\vdash_p\langle$*mark-guards f c1*,*s*$\rangle$ =*n*$\Rightarrow$ *w* **and**
exec-mark-c2: $\Gamma\vdash_p\langle$*mark-guards f c2*,*w*$\rangle$ =*n*$\Rightarrow$ *t*
by (*auto elim*: *execn-elim-cases*)
from *Seq.hyps exec-mark-c1*
obtain w' **where**
exec-c1: $\Gamma\vdash_p\langle$*c1*,*s*$\rangle$ =*n*$\Rightarrow$ w' **and**
w-Fault: *isFault w* $\longrightarrow$ *isFault* w' **and**
w'*-Fault-f*: w' = *Fault f* $\longrightarrow$ w'=*w* **and**
w'*-Fault*: *isFault* w' $\longrightarrow$ *isFault w* **and**
w'*-noFault*: $\neg$ *isFault* w' $\longrightarrow$ w'=*w*
by *blast*
show *?case*
proof (*cases s*)
case (*Fault f*)

```
  with exec-mark have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
  case Stuck
  with exec-mark have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec-mark have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  show ?thesis
  proof (cases isFault w)
    case True
    then obtain f where w': w=Fault f..
    moreover with exec-mark-c2
    have t: t=Fault f
      by (auto dest: execn-Fault-end)
    ultimately show ?thesis
      using Normal w-Fault w'-Fault-f exec-c1
      by (fastforce intro: execn.intros elim: isFaultE)
  next
    case False
    note noFault-w = this
    show ?thesis
    proof (cases isFault w')
      case True
      then obtain f' where w': w'=Fault f'..
      with Normal exec-c1
      have exec: Γ⊢p ⟨Seq c1 c2,s⟩ =n⇒ Fault f'
        by (auto intro: execn.intros)
      from w'-Fault-f w' noFault-w
      have f' ≠ f
        by (cases w) auto
      moreover
      from w' w'-Fault exec-mark-c2 have isFault t
        by (auto dest: execn-Fault-end elim: isFaultE)
      ultimately
      show ?thesis
        using exec
        by auto
    next
```

case *False*
with *w′-noFault* **have** *w′*: $w'=w$ **by** *simp*
from *Seq.hyps exec-mark-c2*
obtain t' **where**
$\Gamma\vdash_p\langle c2,w\rangle =n\Rightarrow t'$ **and**
isFault $t \longrightarrow$ *isFault* t' **and**
$t' =$ *Fault* $f \longrightarrow t'=t$ **and**
isFault $t' \longrightarrow$ *isFault* t **and**
$\neg$ *isFault* $t' \longrightarrow t'=t$
by *blast*
with *Normal exec-c1* w'
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
qed
next
case (*Cond b c1 c2 s n t*)
have *exec-mark*: $\Gamma\vdash_p\langle$*mark-guards f* (*Cond b c1 c2*)$,s\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-mark* **have** $t=$*Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-mark* **have** $t=$*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt s′*)
with *exec-mark* **have** $t=$*Abrupt* s'
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s′*)
show *?thesis*
proof (*cases* $s'\in b$)
case *True*
with *Normal exec-mark*
have $\Gamma\vdash_p\langle$*mark-guards f c1* ,*Normal* $s'\rangle =n\Rightarrow t$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Normal True Cond.hyps* **obtain** t'
where $\Gamma\vdash_p\langle c1,$*Normal* $s'\rangle =n\Rightarrow t'$
isFault $t \longrightarrow$ *isFault* t'

$t' = \mathit{Fault}\ f \longrightarrow t'=t$
$\mathit{isFault}\ t' \longrightarrow \mathit{isFault}\ t$
$\neg\ \mathit{isFault}\ t' \longrightarrow t' = t$
by *blast*
with *Normal True*
show *?thesis*
by (*blast intro*: *execn.intros*)
next
case *False*
with *Normal exec-mark*
have $\Gamma\vdash_p\langle \mathit{mark\text{-}guards}\ f\ c2\ ,\mathit{Normal}\ s'\rangle =n\Rightarrow t$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Normal False Cond.hyps* **obtain** t'
where $\Gamma\vdash_p\langle c2,\mathit{Normal}\ s'\rangle =n\Rightarrow t'$
$\mathit{isFault}\ t \longrightarrow \mathit{isFault}\ t'$
$t' = \mathit{Fault}\ f \longrightarrow t'=t$
$\mathit{isFault}\ t' \longrightarrow \mathit{isFault}\ t$
$\neg\ \mathit{isFault}\ t' \longrightarrow t' = t$
by *blast*
with *Normal False*
show *?thesis*
by (*blast intro*: *execn.intros*)
qed
qed
next
case (*While b c s n t*)
have *exec-mark*: $\Gamma\vdash_p\langle \mathit{mark\text{-}guards}\ f\ (\mathit{While}\ b\ c),s\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-mark* **have** $t=\mathit{Fault}\ f$
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-mark* **have** $t=\mathit{Stuck}$
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt s'*)
with *exec-mark* **have** $t=\mathit{Abrupt}\ s'$
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s'*)
{

fix c' r w
assume *exec-c'*: $\Gamma\vdash_p\langle c',r\rangle =n\Rightarrow w$
assume *c'*: c'=*While* b (*mark-guards* f c)
have $\exists w'$. $\Gamma\vdash_p\langle$*While* b c,$r\rangle =n\Rightarrow w' \wedge$ (*isFault* $w \longrightarrow$ *isFault* w') $\wedge$
(w' = *Fault* $f \longrightarrow w'$=w) $\wedge$ (*isFault* $w' \longrightarrow$ *isFault* w) $\wedge$
($\neg$ *isFault* $w' \longrightarrow w'$=w)
using *exec-c'* *c'*
proof (*induct*)
case (*WhileTrue* r b' c'' n u w)
have *eqs*: *While* b' c'' = *While* b (*mark-guards* f c) **by** *fact*
from *WhileTrue.hyps eqs*
have *r-in-b*: $r\in b$ **by** *simp*
from *WhileTrue.hyps eqs*
have *exec-mark-c*: $\Gamma\vdash_p\langle$*mark-guards* f c,*Normal* $r\rangle =n\Rightarrow u$ **by** *simp*
from *WhileTrue.hyps eqs*
have *exec-mark-w*: $\Gamma\vdash_p\langle$*While* b (*mark-guards* f c),$u\rangle =n\Rightarrow w$
by *simp*
show *?case*
proof –
from *WhileTrue.hyps eqs* **have** $\Gamma\vdash_p\langle$*mark-guards* f c,*Normal* $r\rangle =n\Rightarrow u$
by *simp*
with *While.hyps*
obtain u' **where**
exec-c: $\Gamma\vdash_p\langle c$,*Normal* $r\rangle =n\Rightarrow u'$ **and**
u-Fault: *isFault* $u \longrightarrow$ *isFault* u' **and**
u'-Fault-f: u' = *Fault* $f \longrightarrow u'$=u **and**
u'-Fault: *isFault* $u' \longrightarrow$ *isFault* u **and**
u'-noFault: $\neg$ *isFault* $u' \longrightarrow u'$=u
by *blast*
show *?thesis*
proof (*cases isFault* u')
case *False*
with *u'-noFault* **have** *u'*: u'=u **by** *simp*
from *WhileTrue.hyps eqs* **obtain** w' **where**
$\Gamma\vdash_p\langle$*While* b c,$u\rangle =n\Rightarrow w'$
isFault $w \longrightarrow$ *isFault* w'
w' = *Fault* $f \longrightarrow w'$=w
isFault $w' \longrightarrow$ *isFault* w
$\neg$ *isFault* $w' \longrightarrow w' = w$
by *blast*
with *u'* *exec-c* *r-in-b*
show *?thesis*
by (*blast intro*: *execn.WhileTrue*)
next
case *True*
then obtain f' **where** *u'*: u'=*Fault* f'..
with *exec-c* *r-in-b*
have *exec*: $\Gamma\vdash_p\langle$*While* b c,*Normal* $r\rangle =n\Rightarrow$ *Fault* f'
by (*blast intro*: *execn.intros*)

from *True u′-Fault* **have** *isFault u*
by *simp*
then obtain *f* **where** *u*: *u=Fault f*..
with *exec-mark-w* **have** *w=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *exec u′ u u′-Fault-f*
show *?thesis*
by *auto*
qed
qed
next
case (*WhileFalse r b′ c′′ n*)
have *eqs*: *While b′ c′′ = While b* (*mark-guards f c*) **by** *fact*
from *WhileFalse.hyps eqs*
have *r-not-in-b*: $r \notin b$ **by** *simp*
show *?case*
proof −
from *r-not-in-b*
have $\Gamma\vdash_p\langle$*While b c*,*Normal r*$\rangle$ =*n*⇒ *Normal r*
by (*rule execn.WhileFalse*)
thus *?thesis*
by *blast*
qed
qed *auto*
} **note** *hyp-while* = *this*
show *?thesis*
proof (*cases* $s′ \in b$)
case *False*
with *Normal exec-mark*
have *t=s*
by (*auto elim*: *execn-Normal-elim-cases*)
with *Normal False* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *True* **note** *s′-in-b* = *this*
with *Normal exec-mark* **obtain** *r* **where**
exec-mark-c: $\Gamma\vdash_p\langle$*mark-guards f c*,*Normal s′*$\rangle$ =*n*⇒ *r* **and**
exec-mark-w: $\Gamma\vdash_p\langle$*While b* (*mark-guards f c*),*r*$\rangle$ =*n*⇒ *t*
by (*auto elim*: *execn-Normal-elim-cases*)
from *While.hyps exec-mark-c* **obtain** *r′* **where**
exec-c: $\Gamma\vdash_p\langle$*c*,*Normal s′*$\rangle$ =*n*⇒ *r′* **and**
r-Fault: *isFault r* ⟶ *isFault r′* **and**
r′-Fault-f: *r′ = Fault f* ⟶ *r′=r* **and**
r′-Fault: *isFault r′* ⟶ *isFault r* **and**
r′-noFault: ¬ *isFault r′* ⟶ *r′=r*
by *blast*
show *?thesis*
proof (*cases isFault r′*)
case *False*

```
        with r'-noFault have r': r'=r by simp
        from hyp-while exec-mark-w
        obtain t' where
          Γ⊢_p ⟨While b c,r⟩ =n⇒ t'
          isFault t ⟶ isFault t'
          t' = Fault f ⟶ t'=t
          isFault t' ⟶ isFault t
          ¬ isFault t' ⟶ t'=t
          by blast
        with r' exec-c Normal s'-in-b
        show ?thesis
          by (blast intro: execn.intros)
      next
        case True
        then obtain f' where r': r'=Fault f'..
        hence Γ⊢_p ⟨While b c,r'⟩ =n⇒ Fault f'
          by auto
        with Normal s'-in-b exec-c
        have exec: Γ⊢_p ⟨While b c,Normal s'⟩ =n⇒ Fault f'
          by (auto intro: execn.intros)
        from True r'-Fault
        have isFault r
          by simp
        then obtain f where r: r=Fault f..
        with exec-mark-w have t=Fault f
          by (auto dest: execn-Fault-end)
        with Normal exec r' r r'-Fault-f
        show ?thesis
          by auto
      qed
    qed
  qed
next
  case Call thus ?case by auto
next
  case DynCom thus ?case
    by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
  case (Guard f' g c s n t)
  have exec-mark: Γ⊢_p ⟨mark-guards f (Guard f' g c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-mark have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
```

with *exec-mark* **have** t=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt* s')
with *exec-mark* **have** t=*Abrupt* s'
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s')
show *?thesis*
proof (*cases* $s' \in g$)
case *False*
with *Normal exec-mark* **have** t: t=*Fault f*
by (*auto elim*: *execn-Normal-elim-cases*)
from *False*
have $\Gamma \vdash_p \langle$*Guard* f' g c,*Normal* $s'\rangle$ $=n\Rightarrow$ *Fault* f'
by (*blast intro*: *execn.intros*)
with *Normal* t **show** *?thesis*
by *auto*
next
case *True*
with *exec-mark Normal*
have $\Gamma \vdash_p \langle$*mark-guards f c*,*Normal* $s'\rangle$ $=n\Rightarrow$ t
by (*auto elim*: *execn-Normal-elim-cases*)
with *Guard.hyps* **obtain** t' **where**
$\Gamma \vdash_p \langle c$,*Normal* $s'\rangle$ $=n\Rightarrow$ t' **and**
isFault t $\longrightarrow$ *isFault* t' **and**
t' = *Fault f* $\longrightarrow$ $t'=t$ **and**
isFault t' $\longrightarrow$ *isFault* t **and**
$\neg$ *isFault* t' $\longrightarrow$ $t'=t$
by *blast*
with *Normal True*
show *?thesis*
by (*blast intro*: *execn.intros*)
qed
qed
next
case *Throw* **thus** *?case* **by** *auto*
next
case (*Catch c1 c2 s n t*)
have *exec-mark*: $\Gamma \vdash_p \langle$*mark-guards f* (*Catch c1 c2*),$s\rangle$ $=n\Rightarrow$ t **by** *fact*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-mark* **have** t=*Fault f*
by (*auto dest*: *execn-Fault-end*)

```
  with Fault show ?thesis
    by auto
next
  case Stuck
  with exec-mark have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec-mark have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s') note s=this
  with exec-mark have
   Γ⊢p⟨Catch (mark-guards f c1) (mark-guards f c2),Normal s'⟩ =n⇒ t by simp
  thus ?thesis
  proof (cases)
    fix w
    assume exec-mark-c1: Γ⊢p⟨mark-guards f c1,Normal s'⟩ =n⇒ Abrupt w
    assume exec-mark-c2: Γ⊢p⟨mark-guards f c2,Normal w⟩ =n⇒ t
    from exec-mark-c1 Catch.hyps
    obtain w' where
      exec-c1: Γ⊢p⟨c1,Normal s'⟩ =n⇒ w' and
      w'-Fault-f: w' = Fault f ⟶ w'=Abrupt w and
      w'-Fault: isFault w' ⟶ isFault (Abrupt w) and
      w'-noFault: ¬ isFault w' ⟶ w'=Abrupt w
      by fastforce
    show ?thesis
    proof (cases w')
      case (Fault f')
      with Normal exec-c1 have Γ⊢p⟨Catch c1 c2,s⟩ =n⇒ Fault f'
        by (auto intro: execn.intros)
      with w'-Fault Fault show ?thesis
        by auto
    next
      case Stuck
      with w'-noFault have False
        by simp
      thus ?thesis ..
    next
      case (Normal w'')
      with w'-noFault have False by simp thus ?thesis ..
    next
      case (Abrupt w'')
      with w'-noFault have w'': w''=w by simp
      from  exec-mark-c2 Catch.hyps
```

```
        obtain t' where
          Γ⊢p⟨c2,Normal w⟩ =n⇒ t'
          isFault t ⟶ isFault t'
          t' = Fault f ⟶ t'=t
          isFault t' ⟶ isFault t
          ¬ isFault t' ⟶ t'=t
          by blast
        with w'' Abrupt s exec-c1
        show ?thesis
          by (blast intro: execn.intros)
      qed
    next
      assume t: ¬ isAbr t
      assume Γ⊢p⟨mark-guards f c1,Normal s'⟩ =n⇒ t
      with Catch.hyps
      obtain t' where
        exec-c1: Γ⊢p⟨c1,Normal s'⟩ =n⇒ t'  and
        t-Fault: isFault t ⟶ isFault t' and
        t'-Fault-f: t' = Fault f ⟶ t'=t and
        t'-Fault: isFault t' ⟶ isFault t and
        t'-noFault: ¬ isFault t' ⟶ t'=t
        by blast
      show ?thesis
      proof (cases isFault t')
        case True
        then obtain f' where t': t'=Fault f'..
        with exec-c1 have Γ⊢p⟨Catch c1 c2,Normal s'⟩ =n⇒ Fault f'
          by (auto intro: execn.intros)
        with t'-Fault-f t'-Fault t' s show ?thesis
          by auto
      next
        case False
        with t'-noFault have t'=t by simp
        with t exec-c1 s show ?thesis
          by (blast intro: execn.intros)
      qed
    qed
  qed
next
  case (Await b c e s n t)
  have exec-mark:  Γ⊢p ⟨mark-guards f (Await b c e),s⟩ =n⇒ t  by fact
  thus ?case
  proof (cases s)
    case (Fault f)
      with exec-mark have t=s
      by (auto dest: execn-Fault-end)
      thus ?thesis using Fault by auto
  next
    case Stuck
```

have $t = Stuck$
using *exec-mark Stuck execn-Stuck-end* **by** *blast*
thus *?thesis* **using** *Stuck* **by** *auto*
next
case (*Abrupt* s')
with *exec-mark* **have** t=*Abrupt* s'
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s') **note** s=*this*
{
fix c' r w
assume *exec-c'*: $\Gamma\vdash_p\langle c',r\rangle$ $=n\Rightarrow$ w
assume c': c'=*Await* b (*Language.mark-guards* f c) e
have $\exists\, w'$. $\Gamma\vdash_p\langle$*Await* b c e,$r\rangle$ $=n\Rightarrow$ w' $\wedge$ (*isFault* w $\longrightarrow$ *isFault* w') $\wedge$
(w' = *Fault* f $\longrightarrow$ w'=w) $\wedge$ (*isFault* w' $\longrightarrow$ *isFault* w) $\wedge$
($\neg$ *isFault* w' $\longrightarrow$ w'=w)
using *exec-c'* c'
proof (*induct*)
case (*AwaitTrue* r b' $\Gamma 1$ c'' n u)
then have *eqs*: *Await* b' c'' e= *Await* b (*Language.mark-guards* f c) e **by** *auto*
from *AwaitTrue.hyps eqs*
have *r-in-b*: $r\in b$ **by** *simp*
from *AwaitTrue.hyps eqs*
have *exec-mark-c*: $\Gamma 1\vdash\langle$*Language.mark-guards* f c,*Normal* $r\rangle$ $=n\Rightarrow$ u **by** *simp*
from *AwaitTrue.hyps eqs*
have *exec-mark-w*: $\Gamma\vdash_p\langle$*Await* b (*Language.mark-guards* f c) e,*Normal* $r\rangle$ $=n\Rightarrow$ u
proof –
have $\Gamma_{\neg a}\vdash$ $\langle c''$,*Normal* $r\rangle$ $=n\Rightarrow$ u **using** *AwaitTrue.hyps*(*2*) *AwaitTrue.hyps*(*3*) **by** *presburger*
then have $\Gamma\vdash_p$ $\langle$*Await* b' c'' e,*Normal* $r\rangle$ $=n\Rightarrow$ u
by (*fastforce intro*: *AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *execn.AwaitTrue*)
thus *?thesis*
using *eqs* **by** *auto*
qed
show *?case*
proof –
from *AwaitTrue.hyps eqs* **have** $\Gamma 1\vdash\langle$*Language.mark-guards* f c,*Normal* $r\rangle$ $=n\Rightarrow$ u
by *simp*

obtain u' **where**
exec-c: $\Gamma 1\vdash\langle c$,*Normal* $r\rangle$ $=n\Rightarrow$ u' **and**
u-Fault: *isFault* u $\longrightarrow$ *isFault* u' **and**
u'-Fault-f: u' = *Fault* f $\longrightarrow$ u'=u **and**

u'*-Fault*: *isFault* $u' \longrightarrow$ *isFault* u **and**
u'*-noFault*: $\neg$ *isFault* $u' \longrightarrow u'=u$
by (*metis Semantic.isFaultE SemanticCon.isFault-simps*(*3*) *exec-mark-c execn-mark-guards-to-execn*)
show *?thesis*
proof (*cases isFault* u')
case *False*
with u'*-noFault* **have** u': $u'=u$ **by** *simp*
from *AwaitTrue.hyps eqs* **obtain** w' **where**
$\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ r\rangle = n\Rightarrow w'$
isFault $u \longrightarrow$ *isFault* w'
$w' = Fault\ f \longrightarrow w'=u$
isFault $w' \longrightarrow$ *isFault* u
$\neg$ *isFault* $w' \longrightarrow w' = u$
proof −
assume *a1*: $\bigwedge w'.\ [\![\Gamma\vdash_p \langle Await\ b\ c\ e, Normal\ r\rangle = n\Rightarrow w';$
isFault $u \longrightarrow$ *isFault* w';
$w' = Fault\ f \longrightarrow w' = u$; *isFault* $w' \longrightarrow$ *isFault* u;
$\neg$ *isFault* $w' \longrightarrow w' = u]\!] \Longrightarrow thesis$
have $\Gamma_{\neg a}\vdash \langle c, Normal\ r\rangle = n\Rightarrow u'$ **using** *AwaitTrue.hyps*(*2*) *exec-c* **by** *blast*
then have $\Gamma\vdash_p \langle Await\ b\ c\ e, Normal\ r\rangle = n\Rightarrow u'$
by (*fastforce intro*: *exec-c execn.AwaitTrue r-in-b*)
thus *?thesis*
using *a1* u' **by** *blast*
qed
with u' *exec-c r-in-b*
show *?thesis*
by (*blast intro*: *execn.AwaitTrue*)
next
case *True*
then obtain f' **where** u': $u'=Fault\ f'$..
with *exec-c r-in-b*
have *exec*: $\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ r\rangle = n\Rightarrow Fault\ f'$
by (*simp add*: *AwaitTrue.hyps*(*2*) *execn.AwaitTrue*)
from *True* u'*-Fault* **have** *isFault* u
by *simp*
then obtain f **where** u: $u=Fault\ f$..
with *exec-mark-w* **have** $u=Fault\ f$
by (*auto*)
with *exec* u' u u'*-Fault-f*
show *?thesis*
by *auto*
qed
qed
next
case (*AwaitFalse s b*) **thus** *?case* **using** *execn.AwaitFalse* **by** *fastforce*
qed *auto*
} **note** *hyp-await* = *this*

show *?thesis* **using** *exec-mark hyp-await* **by** *auto*
qed
qed

lemma *exec-to-exec-mark-guards*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma\vdash_p\langle mark\text{-}guards\ f\ c,s\rangle \Rightarrow t$
proof $-$
from *exec-to-execn* [*OF exec-c*] **obtain** *n* **where**
$\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$ **..**
from *execn-to-execn-mark-guards* [*OF this t-not-Fault*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-to-exec-mark-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow Fault\ f$
shows $\exists f'.\ \Gamma\vdash_p\langle mark\text{-}guards\ x\ c,s\rangle \Rightarrow Fault\ f'$
proof $-$
from *exec-to-execn* [*OF exec-c*] **obtain** *n* **where**
$\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow Fault\ f$ **..**
from *execn-to-execn-mark-guards-Fault* [*OF this*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-mark-guards-to-exec*:
assumes *exec-mark*: $\Gamma\vdash_p\langle mark\text{-}guards\ f\ c,s\rangle \Rightarrow t$
shows $\exists t'.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge$
$(t' = Fault\ f \longrightarrow t'=t) \wedge$
$(isFault\ t' \longrightarrow isFault\ t) \wedge$
$(\neg\ isFault\ t' \longrightarrow t'=t)$
proof $-$
from *exec-to-execn* [*OF exec-mark*] **obtain** *n* **where**
$\Gamma\vdash_p\langle mark\text{-}guards\ f\ c,s\rangle =n\Rightarrow t$ **..**
from *execn-mark-guards-to-execn* [*OF this*]
show *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

6.7 Lemmas about *LanguageCon.strip-guards*

lemma *execn-to-execn-strip-guards*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma\vdash_p\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow t$

using *exec-c t-not-Fault* [*simplified not-isFault-iff*]
proof *induct*
 case (*AwaitTrue s b* Γ*1 c n t*)
 then have Γ*1*⊢ ⟨*Language.strip-guards F c*,*Normal s*⟩ =*n*⇒ *t*
  by (*meson Semantic.isFaultE execn-to-execn-strip-guards*)
 thus *?case* **by** (*auto intro*:*AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *execn.AwaitTrue*)
qed (*auto intro*: *execn.intros dest*: *noFaultn-startD'*)

lemma *execn-to-execn-strip-guards-Fault*:
 assumes *exec-c*: Γ⊢$_p$⟨*c*,*s*⟩ =*n*⇒ *t*
 shows ⋀*f*. ⟦*t*=*Fault f*; *f* ∉ *F*⟧ ⟹ Γ⊢$_p$⟨*strip-guards F c*,*s*⟩ =*n*⇒ *Fault f*
using *exec-c*
proof (*induct*)
 case *Skip* **thus** *?case* **by** *auto*
next
 case *Guard* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
 case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
 case *FaultProp* **thus** *?case* **by** *auto*
next
 case *Basic* **thus** *?case* **by** *auto*
next
 case *Spec* **thus** *?case* **by** *auto*
next
 case *SpecStuck* **thus** *?case* **by** *auto*
next
 case (*Seq c1 s n w c2 t*)
 have *exec-c1*: Γ⊢$_p$⟨*c1*,*Normal s*⟩ =*n*⇒ *w* **by** *fact*
 have *exec-c2*: Γ⊢$_p$⟨*c2*,*w*⟩ =*n*⇒ *t* **by** *fact*
 have *t*: *t*=*Fault f* **by** *fact*
 have *notinF*: *f* ∉ *F* **by** *fact*
 show *?case*
 proof (*cases w*)
  case (*Fault f'*)
  with *exec-c2 t* **have** *f'*=*f*
   by (*auto dest*: *execn-Fault-end*)
  with *Fault notinF Seq.hyps*
  have Γ⊢$_p$⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *Fault f*
   by *auto*
  moreover have Γ⊢$_p$⟨*strip-guards F c2*,*Fault f*⟩ =*n*⇒ *Fault f*
   by *auto*
  ultimately show *?thesis*
   by (*auto intro*: *execn.intros*)
 next
  case (*Normal s'*)
  with *execn-to-execn-strip-guards* [*OF exec-c1*]
  have *exec-strip-c1*: Γ⊢$_p$⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *w*

by *simp*
with *Seq.hyps t notinF*
have $\Gamma\vdash_p\langle$*strip-guards F c2*,*w*$\rangle$ =*n*$\Rightarrow$ *Fault f*
by *blast*
with *exec-strip-c1* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case (*Abrupt s'*)
with *execn-to-execn-strip-guards* [*OF exec-c1*]
have *exec-strip-c1*: $\Gamma\vdash_p\langle$*strip-guards F c1*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *w*
by *simp*
with *Seq.hyps t notinF*
have $\Gamma\vdash_p\langle$*strip-guards F c2*,*w*$\rangle$ =*n*$\Rightarrow$ *Fault f*
by (*auto intro*: *execn.intros*)
with *exec-strip-c1* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-c2* **have** *t*=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *t* **show** *?thesis* **by** *simp*
qed
next
case *CondTrue* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case *CondFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case (*WhileTrue s b c n w t*)
have *exec-c*: $\Gamma\vdash_p\langle$*c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *w* **by** *fact*
have *exec-w*: $\Gamma\vdash_p\langle$*While b c*,*w*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
have *t*: *t* = *Fault f* **by** *fact*
have *notinF*: $f \notin F$ **by** *fact*
have *s-in-b*: $s \in b$ **by** *fact*
show *?case*
proof (*cases w*)
case (*Fault f'*)
with *exec-w t* **have** *f'*=*f*
by (*auto dest*: *execn-Fault-end*)
with *Fault notinF WhileTrue.hyps*
have $\Gamma\vdash_p\langle$*strip-guards F c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *Fault f*
by *auto*
moreover have $\Gamma\vdash_p\langle$*strip-guards F* (*While b c*),*Fault f*$\rangle$ =*n*$\Rightarrow$ *Fault f*
by *auto*
ultimately show *?thesis*
using *s-in-b* **by** (*auto intro*: *execn.intros*)
next
case (*Normal s'*)
with *execn-to-execn-strip-guards* [*OF exec-c*]
have *exec-strip-c*: $\Gamma\vdash_p\langle$*strip-guards F c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *w*

```
      by simp
    with WhileTrue.hyps t notinF
    have Γ⊢p⟨strip-guards F (While b c),w⟩ =n⇒ Fault f
      by blast
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-strip-guards [OF exec-c]
    have exec-strip-c: Γ⊢p⟨strip-guards F c,Normal s⟩ =n⇒ w
      by simp
    with WhileTrue.hyps t notinF
    have Γ⊢p⟨strip-guards F (While b c),w⟩ =n⇒ Fault f
      by (auto intro: execn.intros)
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-w have t=Stuck
      by (auto dest: execn-Stuck-end)
    with t show ?thesis by simp
  qed
next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
next
  case Call thus ?case by (fastforce intro: execn.intros)
next
  case CallUndefined thus ?case by simp
next
  case StuckProp thus ?case by simp
next
  case DynCom thus ?case by (fastforce intro: execn.intros)
next
  case Throw thus ?case by simp
next
  case AbruptProp thus ?case by simp
next
  case (CatchMatch c1 s n w c2 t)
  have exec-c1: Γ⊢p⟨c1,Normal s⟩ =n⇒ Abrupt w by fact
  have exec-c2: Γ⊢p⟨c2,Normal w⟩ =n⇒ t by fact
  have t: t = Fault f by fact
  have notinF: f ∉ F by fact
  from execn-to-execn-strip-guards [OF exec-c1]
  have exec-strip-c1: Γ⊢p⟨strip-guards F c1,Normal s⟩ =n⇒ Abrupt w
    by simp
  with CatchMatch.hyps t notinF
  have Γ⊢p⟨strip-guards F c2,Normal w⟩ =n⇒ Fault f
    by blast
  with exec-strip-c1 show ?case
```

by (*auto intro*: *execn.intros*)
next
case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
case (*AwaitTrue s b* Γ*1 c n t*)
then have $\Gamma 1\vdash\langle$*Language.strip-guards F c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *Fault f*
by (*simp add*: *execn-to-execn-strip-guards-Fault*)
then have $\Gamma_{\neg a}\vdash$ $\langle$*Language.strip-guards F c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *Fault f* **using** *AwaitTrue.hyps*(*2*) *AwaitTrue.hyps*(*3*) **using** *AwaitTrue.prems*(*1*) **by** *blast*
thus *?case* **by**(*simp add*: *AwaitTrue.hyps*(*1*) *execn.AwaitTrue*)
next
case (*AwaitFalse s b*) **thus** *?case* **by** (*auto simp add*:*execn.AwaitFalse*)
qed

lemma *execn-to-execn-strip-guards'*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle$ =*n*$\Rightarrow$ *t*
assumes *t-not-Fault*: $t \notin$ *Fault* ` *F*
shows $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
proof (*cases t*)
case (*Fault f*)
with *t-not-Fault exec-c* **show** *?thesis*
by (*auto intro*: *execn-to-execn-strip-guards-Fault*)
qed (*insert exec-c*, *auto intro*: *execn-to-execn-strip-guards*)

lemma *execn-strip-guards-to-execn*:
$\bigwedge$*s n t*. $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle$ =*n*$\Rightarrow$ *t*
$\Longrightarrow \exists\, t'$. $\Gamma\vdash_p\langle c,s\rangle$ =*n*$\Rightarrow$ $t' \wedge$
(*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$
($t' \in$ *Fault* ` ($-$ *F*) $\longrightarrow$ t'=*t*) $\wedge$
($\neg$ *isFault t'* $\longrightarrow$ t'=*t*)
proof (*induct c*)
case *Skip* **thus** *?case* **by** *auto*
next
case *Basic* **thus** *?case* **by** *auto*
next
case *Spec* **thus** *?case* **by** *auto*
next
case (*Seq c1 c2 s n t*)
have *exec-strip*: $\Gamma\vdash_p\langle$*strip-guards F* (*Seq c1 c2*),*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
then obtain *w* **where**
exec-strip-c1: $\Gamma\vdash_p\langle$*strip-guards F c1*,*s*$\rangle$ =*n*$\Rightarrow$ *w* **and**
exec-strip-c2: $\Gamma\vdash_p\langle$*strip-guards F c2*,*w*$\rangle$ =*n*$\Rightarrow$ *t*
by (*auto elim*: *execn-elim-cases*)
from *Seq.hyps exec-strip-c1*
obtain *w'* **where**
exec-c1: $\Gamma\vdash_p\langle c1,s\rangle$ =*n*$\Rightarrow$ *w'* **and**
w-Fault: *isFault w* $\longrightarrow$ *isFault w'* **and**
w'-Fault: $w' \in$ *Fault* ` ($-$ *F*) $\longrightarrow$ w'=*w* **and**
w'-noFault: $\neg$ *isFault w'* $\longrightarrow$ w'=*w*

  by *blast*
show *?case*
proof (*cases s*)
  case (*Fault f*)
  with *exec-strip* **have** *t=Fault f*
    by (*auto dest*: *execn-Fault-end*)
  with *Fault* **show** *?thesis*
    by *auto*
next
  case *Stuck*
  with *exec-strip* **have** *t=Stuck*
    by (*auto dest*: *execn-Stuck-end*)
  with *Stuck* **show** *?thesis*
    by *auto*
next
  case (*Abrupt s′*)
  with *exec-strip* **have** *t=Abrupt s′*
    by (*auto dest*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis*
    by *auto*
next
  case (*Normal s′*)
  show *?thesis*
  proof (*cases isFault w*)
    case *True*
    then obtain *f* **where** *w′*: *w=Fault f*..
    moreover with *exec-strip-c2*
    have *t*: *t=Fault f*
      by (*auto dest*: *execn-Fault-end*)
    ultimately show *?thesis*
      using *Normal w-Fault w′-Fault exec-c1*
      by (*fastforce intro*: *execn.intros elim*: *isFaultE*)
  next
    case *False*
    note *noFault-w = this*
    show *?thesis*
    proof (*cases isFault w′*)
      case *True*
      then obtain *f′* **where** *w′*: *w′=Fault f′*..
      with *Normal exec-c1*
      have *exec*: $\Gamma\vdash_p\langle Seq\ c1\ c2,s\rangle =n\Rightarrow Fault\ f'$
        by (*auto intro*: *execn.intros*)
      from *w′-Fault w′ noFault-w*
      have $f' \in F$
        by (*cases w*) *auto*
      with *exec*
      show *?thesis*
        by *auto*
    next

case *False*
with *w′-noFault* **have** *w′*: $w'=w$ **by** *simp*
from *Seq.hyps exec-strip-c2*
obtain t' **where**
$\Gamma\vdash_p\langle c2,w\rangle =n\Rightarrow t'$ **and**
isFault $t \longrightarrow$ *isFault* t' **and**
$t' \in$ *Fault* ` $(-F) \longrightarrow t'=t$ **and**
$\neg$ *isFault* $t' \longrightarrow t'=t$
by *blast*
with *Normal exec-c1* w'
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
qed
next
next
case (*Cond b c1 c2 s n t*)
have *exec-strip*: $\Gamma\vdash_p\langle$*strip-guards F* (*Cond b c1 c2*)$,s\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-strip* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-strip* **have** *t=Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt s′*)
with *exec-strip* **have** *t=Abrupt s′*
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s′*)
show *?thesis*
proof (*cases* $s' \in b$)
case *True*
with *Normal exec-strip*
have $\Gamma\vdash_p\langle$*strip-guards F c1* ,*Normal s′*$\rangle =n\Rightarrow t$
by (*auto elim*: *execn-Normal-elim-cases*)
with *Normal True Cond.hyps* **obtain** t'
where $\Gamma\vdash_p\langle c1,$*Normal s′*$\rangle =n\Rightarrow t'$
isFault $t \longrightarrow$ *isFault* t'

```
          t' ∈ Fault ‘ (−F) ⟶ t'=t
          ¬ isFault t' ⟶ t' = t
        by blast
      with Normal True
      show ?thesis
        by (blast intro: execn.intros)
    next
      case False
      with Normal exec-strip
      have Γ⊢p ⟨strip-guards F c2 ,Normal s'⟩ =n⇒ t
        by (auto elim: execn-Normal-elim-cases)
      with Normal False Cond.hyps obtain t'
        where Γ⊢p ⟨c2,Normal s'⟩ =n⇒ t'
          isFault t ⟶ isFault t'
          t' ∈ Fault ‘ (−F) ⟶ t'=t
          ¬ isFault t' ⟶ t' = t
        by blast
      with Normal False
      show ?thesis
        by (blast intro: execn.intros)
    qed
  qed
next
  case (While b c s n t)
  have exec-strip: Γ⊢p ⟨strip-guards F (While b c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-strip have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-strip have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec-strip have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
    {
      fix c' r w
      assume exec-c': Γ⊢p ⟨c',r⟩ =n⇒ w
```

```
assume c': c'=While b (strip-guards F c)
have ∃ w'. Γ⊢p⟨While b c,r⟩ =n⇒ w' ∧ (isFault w ⟶ isFault w') ∧
            (w' ∈ Fault ` (−F) ⟶ w'=w) ∧
            (¬ isFault w' ⟶ w'=w)
  using exec-c' c'
proof (induct)
  case (WhileTrue r b' c'' n u w)
  have eqs: While b' c'' = While b (strip-guards F c) by fact
  from WhileTrue.hyps eqs
  have r-in-b: r∈b by simp
  from WhileTrue.hyps eqs
  have exec-strip-c: Γ⊢p⟨strip-guards F c,Normal r⟩ =n⇒ u by simp
  from WhileTrue.hyps eqs
  have exec-strip-w: Γ⊢p⟨While b (strip-guards F c),u⟩ =n⇒ w
    by simp
  show ?case
  proof −
    from WhileTrue.hyps eqs have Γ⊢p⟨strip-guards F c,Normal r⟩ =n⇒ u
      by simp
    with While.hyps
    obtain u' where
      exec-c: Γ⊢p⟨c,Normal r⟩ =n⇒ u' and
      u-Fault: isFault u ⟶ isFault u' and
      u'-Fault: u' ∈ Fault ` (−F) ⟶ u'=u and
      u'-noFault: ¬ isFault u' ⟶ u'=u
      by blast
    show ?thesis
    proof (cases isFault u')
      case False
      with u'-noFault have u': u'=u by simp
      from WhileTrue.hyps eqs obtain w' where
        Γ⊢p⟨While b c,u⟩ =n⇒ w'
        isFault w  ⟶ isFault w'
        w' ∈ Fault ` (−F) ⟶ w'=w
        ¬ isFault w' ⟶ w' = w
        by auto
      with u' exec-c r-in-b
      show ?thesis
        by (blast intro: execn.WhileTrue)
    next
      case True
      then obtain f' where u': u'=Fault f'..
      with exec-c r-in-b
      have exec: Γ⊢p⟨While b c,Normal r⟩ =n⇒ Fault f'
        by (blast intro: execn.intros)
      show ?thesis
      proof (cases isFault u)
        case True
        then obtain f where u: u=Fault f..
```

```
          with exec-strip-w have w=Fault f
            by (auto dest: execn-Fault-end)
          with exec u' u u'-Fault
          show ?thesis
            by auto
        next
          case False
          with u'-Fault u' have f' ∈ F
            by (cases u) auto
          with exec show ?thesis
            by auto
        qed
      qed
    qed
  next
    case (WhileFalse r b' c'' n)
    have eqs: While b'  c'' = While b (strip-guards F c) by fact
    from WhileFalse.hyps eqs
    have r-not-in-b: r∉b by simp
    show ?case
    proof −
      from r-not-in-b
      have Γ⊢p⟨While b c,Normal r⟩ =n⇒ Normal r
        by (rule execn.WhileFalse)
      thus ?thesis
        by blast
    qed
  qed auto
} note hyp-while = this
show ?thesis
proof (cases s'∈b)
  case False
  with Normal exec-strip
  have t=s
    by (auto elim: execn-Normal-elim-cases)
  with Normal False show ?thesis
    by (auto intro: execn.intros)
next
  case True note s'-in-b = this
  with Normal exec-strip obtain r where
    exec-strip-c: Γ⊢p⟨strip-guards F c,Normal s'⟩ =n⇒ r and
    exec-strip-w: Γ⊢p⟨While b (strip-guards F c),r⟩ =n⇒ t
    by (auto elim: execn-Normal-elim-cases)
  from While.hyps exec-strip-c obtain r' where
    exec-c: Γ⊢p⟨c,Normal s'⟩ =n⇒ r' and
    r-Fault: isFault r ⟶ isFault r' and
    r'-Fault: r' ∈ Fault ` (−F) ⟶ r'=r and
    r'-noFault: ¬ isFault r' ⟶ r'=r
    by blast
```

show *?thesis*
proof (*cases isFault r′*)
case *False*
with *r′-noFault* **have** *r′*: $r'{=}r$ **by** *simp*
from *hyp-while exec-strip-w*
obtain t' **where**
$\Gamma\vdash_p\langle \mathit{While}\ b\ c{,}r\rangle\ {=}n{\Rightarrow}\ t'$
$\mathit{isFault}\ t \longrightarrow \mathit{isFault}\ t'$
$t' \in \mathit{Fault}\ \text{`}\ ({-}F) \longrightarrow t'{=}t$
$\neg\ \mathit{isFault}\ t' \longrightarrow t'{=}t$
by *blast*
with *r′ exec-c Normal s′-in-b*
show *?thesis*
by (*blast intro*: *execn.intros*)
next
case *True*
then obtain f' **where** *r′*: $r'{=}\mathit{Fault}\ f'$**..**
hence $\Gamma\vdash_p\langle \mathit{While}\ b\ c{,}r'\rangle\ {=}n{\Rightarrow}\ \mathit{Fault}\ f'$
by *auto*
with *Normal s′-in-b exec-c*
have *exec*: $\Gamma\vdash_p\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ s'\rangle\ {=}n{\Rightarrow}\ \mathit{Fault}\ f'$
by (*auto intro*: *execn.intros*)
show *?thesis*
proof (*cases isFault r*)
case *True*
then obtain f **where** *r*: $r{=}\mathit{Fault}\ f$**..**
with *exec-strip-w* **have** $t{=}\mathit{Fault}\ f$
by (*auto dest*: *execn-Fault-end*)
with *Normal exec r′ r r′-Fault*
show *?thesis*
by *auto*
next
case *False*
with *r′-Fault r′* **have** $f' \in F$
by (*cases r*) *auto*
with *Normal exec* **show** *?thesis*
by *auto*
qed
qed
qed
qed
next
case *Call* **thus** *?case* **by** *auto*
next
case *DynCom* **thus** *?case*
by (*fastforce elim*!: *execn-elim-cases intro*: *execn.intros*)
next
case (*Guard f g c s n t*)
have *exec-strip*: $\Gamma\vdash_p\langle \mathit{strip\text{-}guards}\ F\ (\mathit{Guard}\ f\ g\ c){,}s\rangle\ {=}n{\Rightarrow}\ t$ **by** *fact*

show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec-strip* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-strip* **have** *t=Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt s'*)
with *exec-strip* **have** *t=Abrupt s'*
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s'*)
show *?thesis*
proof (*cases* $f \in F$)
case *True*
with *exec-strip Normal*
have *exec-strip-c*: $\Gamma\vdash_p\langle strip\text{-}guards\ F\ c, Normal\ s'\rangle =n\Rightarrow t$
by *simp*
with *Guard.hyps* **obtain** t' **where**
$\Gamma\vdash_p\langle c, Normal\ s'\rangle =n\Rightarrow t'$ **and**
isFault $t \longrightarrow$ *isFault* t' **and**
$t' \in Fault\ `\ (-F) \longrightarrow t'=t$ **and**
$\neg$ *isFault* $t' \longrightarrow t'=t$
by *blast*
with *Normal True*
show *?thesis*
by (*cases* $s' \in g$) (*fastforce intro*: *execn.intros*)+
next
case *False*
note *f-notin-F* = *this*
show *?thesis*
proof (*cases* $s' \in g$)
case *False*
with *Normal exec-strip f-notin-F* **have** *t*: *t=Fault f*
by (*auto elim*: *execn-Normal-elim-cases*)
from *False*
have $\Gamma\vdash_p\langle Guard\ f\ g\ c, Normal\ s'\rangle =n\Rightarrow Fault\ f$
by (*blast intro*: *execn.intros*)
with *False Normal t* **show** *?thesis*
by *auto*

```
        next
          case True
          with exec-strip Normal f-notin-F
          have Γ⊢p⟨strip-guards F c,Normal s'⟩ =n⇒ t
            by (auto elim: execn-Normal-elim-cases)
          with Guard.hyps obtain t' where
            Γ⊢p⟨c,Normal s'⟩ =n⇒ t' and
            isFault t ⟶ isFault t' and
            t' ∈ Fault ` (−F) ⟶ t'=t and
            ¬ isFault t' ⟶ t'=t
            by blast
          with Normal True
          show ?thesis
            by (blast intro: execn.intros)
        qed
      qed
    qed
  next
    case Throw thus ?case by auto
  next
    case (Catch c1 c2 s n t)
    have exec-strip: Γ⊢p⟨strip-guards F (Catch c1 c2),s⟩ =n⇒ t by fact
    show ?case
    proof (cases s)
      case (Fault f)
      with exec-strip have t=Fault f
        by (auto dest: execn-Fault-end)
      with Fault show ?thesis
        by auto
    next
      case Stuck
      with exec-strip have t=Stuck
        by (auto dest: execn-Stuck-end)
      with Stuck show ?thesis
        by auto
    next
      case (Abrupt s')
      with exec-strip have t=Abrupt s'
        by (auto dest: execn-Abrupt-end)
      with Abrupt show ?thesis
        by auto
    next
      case (Normal s') note s=this
      with exec-strip have
       Γ⊢p⟨Catch (strip-guards F c1) (strip-guards F c2),Normal s'⟩ =n⇒ t by simp
      thus ?thesis
      proof (cases)
        fix w
        assume exec-strip-c1: Γ⊢p⟨strip-guards F c1,Normal s'⟩ =n⇒ Abrupt w
```

assume *exec-strip-c2*: $\Gamma\vdash_p\langle$*strip-guards F c2,Normal w*$\rangle$ $=n\Rightarrow$ *t*
from *exec-strip-c1 Catch.hyps*
obtain w' **where**
exec-c1: $\Gamma\vdash_p\langle$*c1,Normal s'*$\rangle$ $=n\Rightarrow$ w' **and**
w'-Fault: $w' \in$ *Fault* ‘ $(-F) \longrightarrow w'$=*Abrupt w* **and**
w'-noFault: $\neg$ *isFault* $w' \longrightarrow w'$=*Abrupt w*
by *blast*
show *?thesis*
proof (*cases* w')
case (*Fault f'*)
with *Normal exec-c1* **have** $\Gamma\vdash_p\langle$*Catch c1 c2,s*$\rangle$ $=n\Rightarrow$ *Fault f'*
by (*auto intro*: *execn.intros*)
with *w'-Fault Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *w'-noFault* **have** *False*
by *simp*
thus *?thesis* **..**
next
case (*Normal w''*)
with *w'-noFault* **have** *False* **by** *simp* **thus** *?thesis* **..**
next
case (*Abrupt w''*)
with *w'-noFault* **have** *w''*: *w''*=*w* **by** *simp*
from *exec-strip-c2 Catch.hyps*
obtain t' **where**
$\Gamma\vdash_p\langle$*c2,Normal w*$\rangle$ $=n\Rightarrow$ t'
isFault t $\longrightarrow$ *isFault* t'
$t' \in$ *Fault* ‘ $(-F) \longrightarrow t'$=*t*
$\neg$ *isFault* $t' \longrightarrow t'$=*t*
by *blast*
with *w'' Abrupt s exec-c1*
show *?thesis*
by (*blast intro*: *execn.intros*)
qed
next
assume *t*: $\neg$ *isAbr t*
assume $\Gamma\vdash_p\langle$*strip-guards F c1,Normal s'*$\rangle$ $=n\Rightarrow$ *t*
with *Catch.hyps*
obtain t' **where**
exec-c1: $\Gamma\vdash_p\langle$*c1,Normal s'*$\rangle$ $=n\Rightarrow$ t' **and**
t-Fault: *isFault t* $\longrightarrow$ *isFault* t' **and**
t'-Fault: $t' \in$ *Fault* ‘ $(-F) \longrightarrow t'$=*t* **and**
t'-noFault: $\neg$ *isFault* $t' \longrightarrow t'$=*t*
by *blast*
show *?thesis*
proof (*cases isFault* t')
case *True*

```
        then obtain f' where t': t'=Fault f'..
        with exec-c1 have Γ⊢p⟨Catch c1 c2,Normal s'⟩ =n⇒ Fault f'
          by (auto intro: execn.intros)
        with t'-Fault t' s show ?thesis
          by auto
      next
        case False
        with t'-noFault have t'=t by simp
        with t exec-c1 s show ?thesis
          by (blast intro: execn.intros)
      qed
    qed
  qed
next
  case (Await b c e s n t)
  have exec-strip: Γ⊢p⟨strip-guards F (Await b c e),s⟩ =n⇒ t by fact
  thus ?case
  proof (cases s)
  case (Fault f)
    with exec-strip have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec-strip have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec-strip have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
     with exec-strip have
      Γ⊢p⟨Await b (Language.strip-guards F c) e,Normal s'⟩ =n⇒ t by simp
      {
      fix c' r w
      assume exec-c': Γ⊢p⟨c',r⟩ =n⇒ w
      assume c': c'=Await b (Language.strip-guards F c) e
      have ∃ w'. Γ⊢p⟨Await b c e,r⟩ =n⇒ w' ∧ (isFault w ⟶ isFault w') ∧
                  (w' ∈ Fault ‘ (−F) ⟶ w'=w) ∧
                  (¬ isFault w' ⟶ w'=w)
        using exec-c' c'
      proof (induct)
        case (AwaitTrue r b' Γ1 c'' n u e)
```

then have *eqs*: *Await* b' c'' *e* = *Await* *b* (*Language.strip-guards F c*) *e* **by** *auto*
from *AwaitTrue.hyps eqs*
have *r-in-b*: $r \in b$ **by** *simp*
from *AwaitTrue.hyps eqs*
have *exec-strip-c*: $\Gamma 1 \vdash \langle$*Language.strip-guards F c*,*Normal r*$\rangle$ $=n\Rightarrow$ *u* **by** *simp*
from *AwaitTrue.hyps eqs*
have *beq*:$b=b'$ **by** *auto*
from *AwaitTrue.hyps eqs beq*
have *exec-c''*: $\Gamma\vdash_p$ $\langle$*Await* b' c'' *e*,*Normal r*$\rangle$ $=n\Rightarrow$ *u* **by** (*simp add*: *execn.AwaitTrue*)
from *AwaitTrue.hyps eqs exec-c''*
have *exec-strip-w*: $\Gamma\vdash_p\langle$*Await b* (*Language.strip-guards F c*) *e*,*Normal r*$\rangle$ $=n\Rightarrow$ *u*
by *simp*
show *?case*
proof −
from *AwaitTrue.hyps eqs* **have** $\Gamma 1\vdash\langle$*Language.strip-guards F c*,*Normal r*$\rangle$ $=n\Rightarrow$ *u*
by *simp*
obtain u' **where**
exec-c: $\Gamma 1\vdash\langle c$,*Normal r*$\rangle$ $=n\Rightarrow$ u' **and**
u-Fault: *isFault u* $\longrightarrow$ *isFault* u' **and**
u'*-Fault*: $u' \in$ *Fault* ‘ $(-F)$ $\longrightarrow$ $u'=u$ **and**
u'*-noFault*: $\neg$ *isFault* u' $\longrightarrow$ $u'=u$
by (*metis Semantic.isFaultE SemanticCon.isFault-simps*(*3*) *exec-strip-c execn-strip-guards-to-execn*)
show *?thesis* **by** (*metis* (*no-types*) *AwaitTrue.hyps*(*2*) *exec-c execn.AwaitTrue r-in-b* u'*-Fault* u'*-noFault*)
qed
next
case (*AwaitFalse s b*) **thus** *?case* **using** *execn.AwaitFalse* **by** *fastforce*
qed *auto*
} **note** *hyp-while* = *this*
thus *?thesis* **using** *Await.prems* **by** *auto*
qed
qed

lemma *noaw-strip-noaw*:
assumes *noawait*:*noawaits* (*LanguageCon.strip-guards F z*)
shows *noawaits z*
using *noawait*
proof (*induct z*)
case *Skip* **then show** *?case* **by** *fastforce*
next
case *Basic* **then show** *?case* **by** *fastforce*
next
case *Spec* **then show** *?case* **by** *fastforce*

next
 case *Seq* **then show** *?case* **by** *fastforce*
next
 case *Cond* **then show** *?case* **by** *simp*
next
 case *While* **then show** *?case* **by** *simp*
next
 case *Call* **then show** *?case* **by** *fastforce*
next
 case *DynCom* **then show** *?case* **by** *fastforce*
next
 case (*Guard f g c*)
 have *noawaits* (*LanguageCon.strip-guards F c*)
 proof (*cases f*∈*F*)
  case *True* **show** *?thesis* **using** *Guard.prems True* **by** *force*
 next
  case *False* **thus** *?thesis*
  using *strip-guards-simps*(*9*) *noawaits.simps*(*9*) *Guard.prems*
  by *fastforce*
 qed
 thus *?case*
  by (*simp add*: *Guard.hyps*)
next
 case (*Throw*) **then show** *?case* **by** *fastforce*
next
 case (*Catch*) **then show** *?case* **by** *fastforce*
qed *fastforce*

lemma *await-strip-noaw-z-F*:¬ *noawaits* (*LanguageCon.strip-guards F z*)
  $\Longrightarrow$ *noawaits z* $\Longrightarrow$ *P*
proof (*induct z*)
 case *Skip* **thus** *?case* **by** *auto*
next
 case *Basic* **then show** *?case* **by** *fastforce*
next
 case *Spec* **then show** *?case* **by** *fastforce*
next
 case *Seq* **then show** *?case* **by** *fastforce*
next
 case *Cond* **then show** *?case* **by** *fastforce*
next
 case *While* **then show** *?case* **by** *fastforce*
next
 case *Call* **then show** *?case* **by** *fastforce*
next
 case *DynCom* **then show** *?case* **by** *fastforce*
next
 case (*Guard f g c*)
 then have *noawaits c* **using** *Guard.prems*(*2*) **by** *auto*

have ¬ *noawaits* (*LanguageCon.strip-guards F c*)
proof (*cases f*∈*F*)
  case *True* **thus** *?thesis* **using** *Guard.prems* **by** *force*
next
  case *False* **thus** *?thesis*
  using *strip-guards-simps*(*9*) *noawaits.simps*(*9*) *Guard.prems*
  by *fastforce*
qed
thus *?thesis*
  using *Guard.hyps* ‹*noawaits c*› **by** *blast*
next
  case (*Throw*) **then show** *?case* **by** *fastforce*
next
  case (*Catch*) **then show** *?case* **by** *fastforce*
qed *fastforce*

lemma *strip-eq*: $(strip\ F\ \Gamma)_{\neg a}$ = *Language.strip F* $(\Gamma_{\neg a})$
unfolding *Language.strip-def LanguageCon.strip-def no-await-body-def*
apply *rule*
apply (*split option.split*)
apply *auto*
apply (*simp add*: *no-await-strip-guards-eq*)
apply (*rule noaw-strip-noaw*, *assumption*)
apply (*rule await-strip-noaw-z-F*)
by *assumption*

lemma *execn-strip-to-execn*:
  assumes *exec-strip*: $(strip\ F\ \Gamma)\vdash_p\langle c,s\rangle = n\Rightarrow t$
  shows $\exists\, t'.\ \Gamma\vdash_p\langle c,s\rangle = n\Rightarrow t' \wedge$
    $(isFault\ t \longrightarrow isFault\ t') \wedge$
    $(t' \in Fault\ `\ (-\ F) \longrightarrow t'{=}t) \wedge$
    $(\neg\ isFault\ t' \longrightarrow t'{=}t)$
using *exec-strip*
proof (*induct*)
  case *Skip* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *Guard* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *GuardFault* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *FaultProp* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *Basic* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *Spec* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *SpecStuck* **thus** *?case* **by** (*blast intro*: *execn.intros*)
next
  case *Seq* **thus** *?case* **by** (*blast intro*: *execn.intros elim*: *isFaultE*)

```
next
  case CondTrue thus ?case by (blast intro: execn.intros)
next
  case CondFalse thus ?case by (blast intro: execn.intros)
next
  case WhileTrue thus ?case by (blast intro: execn.intros elim: isFaultE)
next
  case WhileFalse thus ?case by (blast intro: execn.intros)
next
  case Call thus ?case
    by simp (blast intro: execn.intros dest: execn-strip-guards-to-execn)
next
  case CallUndefined thus ?case
    by simp (blast intro: execn.intros)
next
  case StuckProp thus ?case
    by blast
next
  case DynCom thus ?case by (blast intro: execn.intros)
next
  case Throw thus ?case by (blast intro: execn.intros)
next
  case AbruptProp thus ?case by (blast intro: execn.intros)
next
  case (CatchMatch c1 s n r c2 t)
  then obtain r' t' where
    exec-c1: Γ⊢p⟨c1,Normal s⟩ =n⇒ r'  and
    r'-Fault: r' ∈ Fault ` (−F) ⟶ r' = Abrupt r and
    r'-noFault: ¬ isFault r' ⟶ r' = Abrupt r and
    exec-c2: Γ⊢p⟨c2,Normal r⟩ =n⇒ t' and
    t-Fault: isFault t ⟶ isFault t' and
    t'-Fault: t' ∈ Fault ` (−F) ⟶ t' = t and
    t'-noFault: ¬ isFault t' ⟶ t' = t
    by blast
  show ?case
  proof (cases isFault r')
    case True
    then obtain f' where r': r'=Fault f'..
    with exec-c1 have Γ⊢p⟨Catch c1 c2,Normal s⟩ =n⇒ Fault f'
      by (auto intro: execn.intros)
    with r' r'-Fault show ?thesis
      by (auto intro: execn.intros)
  next
    case False
    with r'-noFault have r'=Abrupt r by simp
    with exec-c1 exec-c2 t-Fault t'-noFault t'-Fault
    show ?thesis
      by (blast intro: execn.intros)
  qed
```

next
  case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros elim*: *isFaultE*)
next
  case *AwaitTrue* **thus** *?case*
    by (*metis Semantic.isFaultE SemanticCon.isFault-simps*(*3*) *execn.AwaitTrue execn-strip-to-execn strip-eq*)
next
  case *AwaitFalse* **thus** *?case* **by** (*fastforce intro*: *execn.intros*(*14*))
qed

lemma *exec-strip-guards-to-exec*:
  assumes *exec-strip*: $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
  shows $\exists\, t'.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t' \wedge$
    (*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$
    ($t' \in$ *Fault* ‘ ($-F$) $\longrightarrow t'=t$) $\wedge$
    ($\neg$ *isFault t'* $\longrightarrow t'=t$)
proof −
  from *exec-strip* **obtain** *n* **where**
    execn-strip: $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle =n\Rightarrow t$
    by (*auto simp add*: *exec-iff-execn*)
  then obtain t' **where**
    $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t'$
    isFault t $\longrightarrow$ *isFault t'* $t' \in$ *Fault* ‘ ($-F$) $\longrightarrow t'=t$ $\neg$ *isFault t'* $\longrightarrow t'=t$
    by (*blast dest*: *execn-strip-guards-to-execn*)
  thus *?thesis*
    by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-strip-to-exec*:
  assumes *exec-strip*: *strip F* $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
  shows $\exists\, t'.\ \Gamma\vdash_p\langle c,s\rangle \Rightarrow t' \wedge$
    (*isFault t* $\longrightarrow$ *isFault t'*) $\wedge$
    ($t' \in$ *Fault* ‘ ($-F$) $\longrightarrow t'=t$) $\wedge$
    ($\neg$ *isFault t'* $\longrightarrow t'=t$)
proof −
  from *exec-strip* **obtain** *n* **where**
    execn-strip: *strip F* $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
    by (*auto simp add*: *exec-iff-execn*)
  then obtain t' **where**
    $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t'$
    isFault t $\longrightarrow$ *isFault t'* $t' \in$ *Fault* ‘ ($-F$) $\longrightarrow t'=t$ $\neg$ *isFault t'* $\longrightarrow t'=t$
    by (*blast dest*: *execn-strip-to-execn*)
  thus *?thesis*
    by (*blast intro*: *execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards*:
 assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$

assumes *t-not-Fault*: $\neg$ *isFault t*
shows $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
proof $-$
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle =n\Rightarrow t$
by (*rule execn-to-execn-strip-guards*)
thus $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards'*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
shows $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
proof $-$
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle =n\Rightarrow t$
by (*rule execn-to-execn-strip-guards'*)
thus $\Gamma\vdash_p\langle$*strip-guards F c*,*s*$\rangle \Rightarrow t$
by (*rule execn-to-exec*)
qed

lemma *execn-to-execn-strip*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows *strip F* $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
using *exec-c t-not-Fault*
proof (*induct*)
case (*Call p bdy s n s'*)
have *bdy*: Γ *p* = *Some bdy* **by** *fact*
from *Call* **have** *strip F* $\Gamma\vdash_p\langle$*bdy*,*Normal s*$\rangle =n\Rightarrow s'$
by *blast*
from *execn-to-execn-strip-guards* [*OF this*] *Call*
have *strip F* $\Gamma\vdash_p\langle$*strip-guards F bdy*,*Normal s*$\rangle =n\Rightarrow s'$
by *simp*
moreover from *bdy* **have** (*strip F* Γ) *p* = *Some* (*strip-guards F bdy*)
by *simp*
ultimately
show *?case*
by (*blast intro*: *execn.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*auto intro*: *execn.CallUndefined*)
next
case (*AwaitTrue*) **thus** *?case* **using** *execn-to-execn-strip* **by** (*metis Semantic.isFaultE SemanticCon.isFault-simps*(*3*) *execn.AwaitTrue strip-eq*)

qed (*auto intro*: *execn.intros dest*: *noFaultn-startD′ simp add*: *not-isFault-iff*)

lemma *execn-to-execn-strip′*:
 assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
 assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
 shows *strip F* $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
using *exec-c t-not-Fault*
proof (*induct*)
  case (*Call p bdy s n s′*)
  have *bdy*: Γ *p* = *Some bdy* **by** *fact*
  from *Call* **have** *strip F* $\Gamma\vdash_p\langle bdy,Normal\ s\rangle =n\Rightarrow s'$
    by *blast*
  from *execn-to-execn-strip-guards′* [*OF this*] *Call*
  have *strip F* $\Gamma\vdash_p\langle strip\text{-}guards\ F\ bdy,Normal\ s\rangle =n\Rightarrow s'$
    by *simp*
  moreover from *bdy* **have** (*strip F* Γ) *p* = *Some* (*strip-guards F bdy*)
    by *simp*
  ultimately
  show *?case*
    by (*blast intro*: *execn.intros*)
next
  case *CallUndefined* **thus** *?case* **by** (*auto intro*: *execn.CallUndefined*)
next
  case (*Seq c1 s n s′ c2 t*)
  show *?case*
  proof (*cases isFault s′*)
    case *False*
    with *Seq* **show** *?thesis*
      by (*auto intro*: *execn.intros simp add*: *not-isFault-iff*)
  next
    case *True*
    then obtain *f′* **where** *s′*: *s′=Fault f′* **by** (*auto simp add*: *isFault-def*)
    with *Seq* **obtain** *t=Fault f′* **and** $f' \notin F$
      by (*force dest*: *execn-Fault-end*)
    with *Seq s′* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  qed
next
  case (*WhileTrue b c s n s′ t*)
  show *?case*
  proof (*cases isFault s′*)
    case *False*
    with *WhileTrue* **show** *?thesis*
      by (*auto intro*: *execn.intros simp add*: *not-isFault-iff*)
  next
    case *True*
    then obtain *f′* **where** *s′*: *s′=Fault f′* **by** (*auto simp add*: *isFault-def*)
    with *WhileTrue* **obtain** *t=Fault f′* **and** $f' \notin F$
      by (*force dest*: *execn-Fault-end*)

with *WhileTrue s'* **show** *?thesis*
by (*auto intro*: *execn.intros*)
qed
next
case (*AwaitTrue*) **thus** *?case* **by** (*metis execn.AwaitTrue strip-eq execn-to-execn-strip'*)
qed (*auto intro*: *execn.intros*)

lemma *exec-to-exec-strip*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $\neg$ *isFault t*
shows *strip F* $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have *strip F* $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*rule execn-to-execn-strip*)
thus *strip F* $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip'*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $t \notin$ *Fault* ‘ *F*
shows *strip F* $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *this t-not-Fault*
have *strip F* $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*rule execn-to-execn-strip'*)
thus *strip F* $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
by (*rule execn-to-exec*)
qed

lemma *exec-to-exec-strip-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow$ *Fault f*
assumes *f-notin-F*: $f \notin F$
shows$\Gamma\vdash_p\langle$*strip-guards F c*,$s\rangle \Rightarrow$ *Fault f*
proof −
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow$*Fault f*
by (*auto simp add*: *exec-iff-execn*)
from *execn-to-execn-strip-guards-Fault* [*OF this - f-notin-F*]
have $\Gamma\vdash_p\langle$*strip-guards F c*,$s\rangle =n\Rightarrow$ *Fault f*
by *simp*
thus $\Gamma\vdash_p\langle$*strip-guards F c*,$s\rangle \Rightarrow$ *Fault f*
by (*rule execn-to-exec*)
qed

6.8 Lemmas about $c_1 \cap_g c_2$

lemma *inter-guards-execn-Normal-noFault*:
$\bigwedge c\ c2\ s\ t\ n.\ [\![(c1 \cap_{gs} c2) = Some\ c;\ \Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t;\ \neg\ isFault\ t]\!]$
$\Longrightarrow \Gamma\vdash_p\langle c1,Normal\ s\rangle =n\Rightarrow t \wedge \Gamma\vdash_p\langle c2,Normal\ s\rangle =n\Rightarrow t$
proof (*induct c1*)
case *Skip*
have $(Skip \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *c2*: *c2=Skip* **and** *c*: *c=Skip*
by (*simp add*: *inter-guards-Skip*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **have** *t=Normal s*
by (*auto elim*: *execn-Normal-elim-cases*)
with *Skip c2*
show *?case*
by (*auto intro*: *execn.intros*)
next
case (*Basic f e*)
have $(Basic\ f\ e \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *c2*: *c2=Basic f e* **and** *c*: *c=Basic f e*
by (*simp add*: *inter-guards-Basic*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **have** *t=Normal* (*f s*)
by (*auto elim*: *execn-Normal-elim-cases*)
with *Basic c2*
show *?case*
by (*auto intro*: *execn.intros*)
next
case (*Spec r e*)
have $(Spec\ r\ e \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *c2*: *c2=Spec r e* **and** *c*: *c=Spec r e*
by (*simp add*: *inter-guards-Spec*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **have** $\Gamma\vdash_p\langle Spec\ r\ e,Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
from *this Spec c2* **show** *?case*
by (*cases*) (*auto intro*: *execn.intros*)
next
case (*Seq a1 a2*)
have *noFault*: ¬ *isFault t* **by** *fact*
have $(Seq\ a1\ a2 \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *b1 b2 d1 d2* **where**
c2: *c2=Seq b1 b2* **and**
d1: $(a1 \cap_{gs} b1) = Some\ d1$ **and** *d2*: $(a2 \cap_{gs} b2) = Some\ d2$ **and**
c: *c=Seq d1 d2*
by (*auto simp add*: *inter-guards-Seq*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **obtain** s' **where**
exec-d1: $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow s'$ **and**
exec-d2: $\Gamma\vdash_p\langle d2,s'\rangle =n\Rightarrow t$
by (*auto elim*: *execn-Normal-elim-cases*)

show *?case*
proof (*cases s'*)
case (*Fault f'*)
with *exec-d2* **have** t=*Fault f'*
by (*auto intro*: *execn-Fault-end*)
with *noFault* **show** *?thesis* **by** *simp*
next
case (*Normal s''*)
with *d1 exec-d1 Seq.hyps*
obtain
$\Gamma\vdash_p\langle a1, Normal\ s\rangle =n\Rightarrow Normal\ s''$ **and** $\Gamma\vdash_p\langle b1, Normal\ s\rangle =n\Rightarrow Normal\ s''$
by *auto*
moreover
from *Normal d2 exec-d2 noFault Seq.hyps*
obtain $\Gamma\vdash_p\langle a2, Normal\ s''\rangle =n\Rightarrow t$ **and** $\Gamma\vdash_p\langle b2, Normal\ s''\rangle =n\Rightarrow t$
by *auto*
ultimately
show *?thesis*
using *Normal c2* **by** (*auto intro*: *execn.intros*)
next
case (*Abrupt s''*)
with *exec-d2* **have** t=*Abrupt s''*
by (*auto simp add*: *execn-Abrupt-end*)
moreover
from *Abrupt d1 exec-d1 Seq.hyps*
obtain $\Gamma\vdash_p\langle a1, Normal\ s\rangle =n\Rightarrow Abrupt\ s''$ **and** $\Gamma\vdash_p\langle b1, Normal\ s\rangle =n\Rightarrow Abrupt\ s''$
by *auto*
moreover
obtain
$\Gamma\vdash_p\langle a2, Abrupt\ s''\rangle =n\Rightarrow Abrupt\ s''$ **and** $\Gamma\vdash_p\langle b2, Abrupt\ s''\rangle =n\Rightarrow Abrupt\ s''$
by *auto*
ultimately
show *?thesis*
using *Abrupt c2* **by** (*auto intro*: *execn.intros*)
next
case *Stuck*
with *exec-d2* **have** t=*Stuck*
by (*auto simp add*: *execn-Stuck-end*)
moreover
from *Stuck d1 exec-d1 Seq.hyps*
obtain $\Gamma\vdash_p\langle a1, Normal\ s\rangle =n\Rightarrow Stuck$ **and** $\Gamma\vdash_p\langle b1, Normal\ s\rangle =n\Rightarrow Stuck$
by *auto*
moreover
obtain
$\Gamma\vdash_p\langle a2, Stuck\rangle =n\Rightarrow Stuck$ **and** $\Gamma\vdash_p\langle b2, Stuck\rangle =n\Rightarrow Stuck$
by *auto*
ultimately
show *?thesis*

```
      using Stuck c2 by (auto intro: execn.intros)
  qed
next
  case (Cond b t1 e1)
  have noFault: ¬ isFault t by fact
  have (Cond b t1 e1 ∩_gs c2) = Some c by fact
  then obtain t2 e2 t3 e3 where
    c2: c2=Cond b t2 e2 and
    t3: (t1 ∩_gs t2) = Some t3 and
    e3: (e1 ∩_gs e2) = Some e3 and
    c: c=Cond b t3 e3
    by (auto simp add: inter-guards-Cond)
  have Γ⊢_p⟨c,Normal s⟩ =n⇒ t by fact
  with c have Γ⊢_p⟨Cond b t3 e3,Normal s⟩ =n⇒ t
    by simp
  then show ?case
  proof (cases)
    assume s-in-b: s∈b
    assume Γ⊢_p⟨t3,Normal s⟩ =n⇒ t
    with Cond.hyps t3 noFault
    obtain Γ⊢_p⟨t1,Normal s⟩ =n⇒ t Γ⊢_p⟨t2,Normal s⟩ =n⇒ t
      by auto
    with s-in-b c2 show ?thesis
      by (auto intro: execn.intros)
  next
    assume s-notin-b: s∉b
    assume Γ⊢_p⟨e3,Normal s⟩ =n⇒ t
    with Cond.hyps e3 noFault
    obtain Γ⊢_p⟨e1,Normal s⟩ =n⇒ t Γ⊢_p⟨e2,Normal s⟩ =n⇒ t
      by auto
    with s-notin-b c2 show ?thesis
      by (auto intro: execn.intros)
  qed
next
  case (While b bdy1)
  have noFault: ¬ isFault t by fact
  have (While b bdy1 ∩_gs c2) = Some c by fact
  then obtain bdy2 bdy where
    c2: c2=While b bdy2 and
    bdy: (bdy1 ∩_gs bdy2) = Some bdy and
    c: c=While b bdy
    by (auto simp add: inter-guards-While)
  have exec-c: Γ⊢_p⟨c,Normal s⟩ =n⇒ t by fact
  {
    fix s t n w w1 w2
    assume exec-w: Γ⊢_p⟨w,Normal s⟩ =n⇒ t
    assume w: w=While b bdy
    assume noFault: ¬ isFault t
    from exec-w w noFault
```

```
have Γ⊢p⟨While b bdy1,Normal s⟩ =n⇒ t ∧
     Γ⊢p⟨While b bdy2,Normal s⟩ =n⇒ t
proof (induct)
  prefer 10
  case (WhileTrue s b' bdy' n s' s'')
  have eqs: While b'  bdy' = While b bdy by fact
  from WhileTrue have s-in-b: s ∈ b by simp
  have noFault-s'': ¬ isFault s''  by fact
  from WhileTrue
  have exec-bdy: Γ⊢p⟨bdy,Normal s⟩ =n⇒ s' by simp
  from WhileTrue
  have exec-w: Γ⊢p⟨While b bdy,s'⟩ =n⇒ s'' by simp
  show ?case
  proof (cases s')
    case (Fault f)
    with exec-w have s''=Fault f
      by (auto intro: execn-Fault-end)
    with noFault-s'' show ?thesis by simp
  next
    case (Normal s''')
    with exec-bdy bdy While.hyps
    obtain Γ⊢p⟨bdy1,Normal s⟩ =n⇒ Normal s'''
           Γ⊢p⟨bdy2,Normal s⟩ =n⇒ Normal s'''
      by auto
    moreover
    from Normal WhileTrue
    obtain
      Γ⊢p⟨While b bdy1,Normal s'''⟩ =n⇒ s''
      Γ⊢p⟨While b bdy2,Normal s'''⟩ =n⇒ s''
      by simp
    ultimately show ?thesis
      using s-in-b Normal
      by (auto intro: execn.intros)
  next
    case (Abrupt s''')
    with exec-bdy bdy While.hyps
    obtain Γ⊢p⟨bdy1,Normal s⟩ =n⇒ Abrupt s'''
           Γ⊢p⟨bdy2,Normal s⟩ =n⇒ Abrupt s'''
      by auto
    moreover
    from Abrupt WhileTrue
    obtain
      Γ⊢p⟨While b bdy1,Abrupt s'''⟩ =n⇒ s''
      Γ⊢p⟨While b bdy2,Abrupt s'''⟩ =n⇒ s''
      by simp
    ultimately show ?thesis
      using s-in-b Abrupt
      by (auto intro: execn.intros)
  next
```

case *Stuck*
with *exec-bdy bdy While.hyps*
obtain $\Gamma\vdash_p\langle bdy1, Normal\ s\rangle =n\Rightarrow Stuck$
$\Gamma\vdash_p\langle bdy2, Normal\ s\rangle =n\Rightarrow Stuck$
by *auto*
moreover
from *Stuck WhileTrue*
obtain
$\Gamma\vdash_p\langle While\ b\ bdy1, Stuck\rangle =n\Rightarrow s''$
$\Gamma\vdash_p\langle While\ b\ bdy2, Stuck\rangle =n\Rightarrow s''$
by *simp*
ultimately show *?thesis*
using *s-in-b Stuck*
by (*auto intro*: *execn.intros*)
qed
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.intros*)
qed (*simp-all*)
}
with *this* [*OF exec-c c noFault*] *c2*
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** (*simp add*: *inter-guards-Call*)
next
case (*DynCom f1*)
have *noFault*: $\neg$ *isFault t* **by** *fact*
have $(DynCom\ f1 \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *f2 f* **where**
c2: *c2=DynCom f2* **and**
f-defined: $\forall s.\ ((f1\ s) \cap_{gs} (f2\ s)) \neq None$ **and**
c: $c{=}DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_{gs} (f2\ s)))$
by (*auto simp add*: *inter-guards-DynCom*)
have $\Gamma\vdash_p\langle c, Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **have** $\Gamma\vdash_p\langle DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_{gs} (f2\ s))), Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
then show *?case*
proof (*cases*)
assume *exec-f*: $\Gamma\vdash_p\langle the\ (f1\ s \cap_{gs} f2\ s), Normal\ s\rangle =n\Rightarrow t$
from *f-defined* **obtain** *f* **where** $(f1\ s \cap_{gs} f2\ s) = Some\ f$
by *auto*
with *DynCom.hyps this exec-f c2 noFault*
show *?thesis*
using *execn.DynCom* **by** *fastforce*
qed
next
case *Guard* **thus** *?case*
by (*fastforce elim*: *execn-Normal-elim-cases intro*: *execn.intros*
simp add: *inter-guards-Guard*)

next
  case *Throw* **thus** *?case*
    by (*fastforce elim*: *execn-Normal-elim-cases*
      simp add: *inter-guards-Throw*)
next
  case (*Catch a1 a2*)
  have *noFault*: ¬ *isFault t* **by** *fact*
  have (*Catch a1 a2* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
  then obtain *b1 b2 d1 d2* **where**
    c2: *c2*=*Catch b1 b2* **and**
    d1: (*a1* $\cap_{gs}$ *b1*) = *Some d1* **and** *d2*: (*a2* $\cap_{gs}$ *b2*) = *Some d2* **and**
    c: *c*=*Catch d1 d2*
    by (*auto simp add*: *inter-guards-Catch*)
  have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
  with *c* **have** $\Gamma\vdash_p\langle Catch\ d1\ d2,Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
  then show *?case*
  proof (*cases*)
    fix *s'*
    assume $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
    with *d1 Catch.hyps*
    obtain $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$ **and** $\Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow Abrupt$
s'
      by *auto*
    moreover
    assume $\Gamma\vdash_p\langle d2,Normal\ s'\rangle =n\Rightarrow t$
    with *d2 Catch.hyps noFault*
    obtain $\Gamma\vdash_p\langle a2,Normal\ s'\rangle =n\Rightarrow t$ **and** $\Gamma\vdash_p\langle b2,Normal\ s'\rangle =n\Rightarrow t$
      by *auto*
    ultimately
    show *?thesis*
      using *c2* **by** (*auto intro*: *execn.intros*)
  next
    assume ¬ *isAbr t*
    moreover
    assume $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow t$
    with *d1 Catch.hyps noFault*
    obtain $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow t$ **and** $\Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow t$
      by *auto*
    ultimately
    show *?thesis*
      using *c2* **by** (*auto intro*: *execn.intros*)
  qed
next
 case (*Await b bdy1 e*)
  have *noFault*: ¬ *isFault t* **by** *fact*
  have (*Await b bdy1 e* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
  then obtain *bdy2 bdy* **where**
    c2: *c2*=*Await b bdy2 e* **and**
    bdy: (*bdy1* $\cap_g$ *bdy2*) = *Some bdy* **and**

c: *c=Await b bdy e*
by (*auto simp add*: *inter-guards-Await*)
have *exec-c*: $\Gamma\vdash_p\langle c, Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
then have $\Gamma\vdash_p\langle Await\ b\ bdy1\ e, Normal\ s\rangle =n\Rightarrow t$
by (*metis Semantic.isFaultE SemanticCon.execn-Normal-elim-cases*(*11*) *SemanticCon.isFault-simps*(*3*) *bdy c execn.AwaitFalse execn.AwaitTrue inter-guards-execn-Normal-noFault noFault*)
thus *?case* **using** *exec-c*
by (*metis Semantic.isFaultE SemanticCon.execn-Normal-elim-cases*(*11*) *SemanticCon.isFault-simps*(*3*) *bdy c execn.AwaitFalse c2 execn.AwaitTrue inter-guards-execn-Normal-noFault noFault*)
qed

lemma *inter-guards-execn-noFault*:
assumes *c*: $(c1 \cap_{gs} c2) = Some\ c$
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
assumes *noFault*: $\neg$ *isFault t*
shows $\Gamma\vdash_p\langle c1,s\rangle =n\Rightarrow t \wedge \Gamma\vdash_p\langle c2,s\rangle =n\Rightarrow t$
proof (*cases s*)
case (*Fault f*)
with *exec-c* **have** *t = Fault f*
by (*auto intro*: *execn-Fault-end*)
with *noFault* **show** *?thesis*
by *simp*
next
case (*Abrupt s'*)
with *exec-c* **have** *t=Abrupt s'*
by (*simp add*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis* **by** *auto*
next
case *Stuck*
with *exec-c* **have** *t=Stuck*
by (*simp add*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis* **by** *auto*
next
case (*Normal s'*)
with *exec-c noFault inter-guards-execn-Normal-noFault* [*OF c*]
show *?thesis*
by *blast*
qed

lemma *inter-guards-exec-noFault*:
assumes *c*: $(c1 \cap_{gs} c2) = Some\ c$
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *noFault*: $\neg$ *isFault t*
shows $\Gamma\vdash_p\langle c1,s\rangle \Rightarrow t \wedge \Gamma\vdash_p\langle c2,s\rangle \Rightarrow t$
proof $-$
from *exec-c* **obtain** *n* **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$

by (*auto simp add*: *exec-iff-execn*)
from *c this noFault*
have $\Gamma\vdash_p\langle c1,s\rangle =n\Rightarrow t \wedge \Gamma\vdash_p\langle c2,s\rangle =n\Rightarrow t$
by (*rule inter-guards-execn-noFault*)
thus *?thesis*
by (*auto intro*: *execn-to-exec*)
qed

lemma *inter-guards-execn-Normal-Fault*:
$\bigwedge c\ c2\ s\ n.\ [\![(c1 \cap_{gs} c2) = Some\ c;\ \Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f]\!]$
$\Longrightarrow (\Gamma\vdash_p\langle c1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash_p\langle c2,Normal\ s\rangle =n\Rightarrow Fault\ f)$
proof (*induct c1*)
case *Skip* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Skip*)
next
case (*Basic f*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Basic*)
next
case (*Spec r*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Spec*)
next
case (*Seq a1 a2*)
have $(Seq\ a1\ a2 \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *b1 b2 d1 d2* **where**
c2: *c2=Seq b1 b2* **and**
d1: $(a1 \cap_{gs} b1) = Some\ d1$ **and** *d2*: $(a2 \cap_{gs} b2) = Some\ d2$ **and**
c: *c=Seq d1 d2*
by (*auto simp add*: *inter-guards-Seq*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
with *c* **obtain** s' **where**
exec-d1: $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow s'$ **and**
exec-d2: $\Gamma\vdash_p\langle d2,s'\rangle =n\Rightarrow Fault\ f$
by (*auto elim*: *execn-Normal-elim-cases*)
show *?case*
proof (*cases s'*)
case (*Fault f'*)
with *exec-d2* **have** *f'=f*
by (*auto dest*: *execn-Fault-end*)
with *Fault d1 exec-d1*
have $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*auto dest*: *Seq.hyps*)
thus *?thesis*
proof (*cases rule*: *disjE* [*consumes 1*])
assume $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f$
hence $\Gamma\vdash_p\langle Seq\ a1\ a2,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*auto intro*: *execn.intros*)
thus *?thesis*
by *simp*
next
assume $\Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
hence $\Gamma\vdash_p\langle Seq\ b1\ b2,Normal\ s\rangle =n\Rightarrow Fault\ f$

```
        by (auto intro: execn.intros)
      with c2 show ?thesis
        by simp
    qed
  next
    case Abrupt with exec-d2 show ?thesis by (auto dest: execn-Abrupt-end)
  next
    case Stuck with exec-d2 show ?thesis by (auto dest: execn-Stuck-end)
  next
    case (Normal s'')
    with inter-guards-execn-noFault [OF d1 exec-d1] obtain
      exec-a1: Γ⊢p⟨a1,Normal s⟩ =n⇒ Normal s'' and
      exec-b1: Γ⊢p⟨b1,Normal s⟩ =n⇒ Normal s''
      by simp
    moreover from d2 exec-d2 Normal
    have Γ⊢p⟨a2,Normal s''⟩ =n⇒ Fault f ∨ Γ⊢p⟨b2,Normal s''⟩ =n⇒ Fault f
      by (auto dest: Seq.hyps)
    ultimately show ?thesis
      using c2 by (auto intro: execn.intros)
  qed
next
  case (Cond b t1 e1)
  have (Cond b t1 e1 ∩gs c2) = Some c by fact
  then obtain t2 e2 t e where
    c2: c2=Cond b t2 e2 and
    t: (t1 ∩gs t2) = Some t and
    e: (e1 ∩gs e2) = Some e and
    c: c=Cond b t e
    by (auto simp add: inter-guards-Cond)
  have Γ⊢p⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢p⟨Cond b t e,Normal s⟩ =n⇒ Fault f by simp
  thus ?case
  proof (cases)
    assume s ∈ b
    moreover assume Γ⊢p⟨t,Normal s⟩ =n⇒ Fault f
    with t have Γ⊢p⟨t1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢p⟨t2,Normal s⟩ =n⇒ Fault
f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  next
    assume s ∉ b
    moreover assume Γ⊢p⟨e,Normal s⟩ =n⇒ Fault f
    with e have Γ⊢p⟨e1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢p⟨e2,Normal s⟩ =n⇒ Fault
f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  qed
next
  case (While b bdy1)
```

have (*While b bdy1* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
then obtain *bdy2 bdy* **where**
c2: *c2*=*While b bdy2* **and**
bdy: (*bdy1* $\cap_{gs}$ *bdy2*) = *Some bdy* **and**
c: *c*=*While b bdy*
by (*auto simp add*: *inter-guards-While*)
have *exec-c*: $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
{
fix *s t n w w1 w2*
assume *exec-w*: $\Gamma\vdash_p\langle w,Normal\ s\rangle =n\Rightarrow t$
assume *w*: *w*=*While b bdy*
assume *Fault*: *t*=*Fault f*
from *exec-w w Fault*
have $\Gamma\vdash_p\langle While\ b\ bdy1,Normal\ s\rangle =n\Rightarrow Fault\ f\vee$
$\Gamma\vdash_p\langle While\ b\ bdy2,Normal\ s\rangle =n\Rightarrow Fault\ f$
proof (*induct*)
case (*WhileTrue s b′ bdy′ n s′ s″*)
have *eqs*: *While b′ bdy′* = *While b bdy* **by** *fact*
from *WhileTrue* **have** *s-in-b*: $s \in b$ **by** *simp*
have *Fault-s″*: *s″*=*Fault f* **by** *fact*
from *WhileTrue*
have *exec-bdy*: $\Gamma\vdash_p\langle bdy,Normal\ s\rangle =n\Rightarrow s'$ **by** *simp*
from *WhileTrue*
have *exec-w*: $\Gamma\vdash_p\langle While\ b\ bdy,s'\rangle =n\Rightarrow s''$ **by** *simp*
show *?case*
proof (*cases s′*)
case (*Fault f′*)
with *exec-w Fault-s″* **have** *f′*=*f*
by (*auto dest*: *execn-Fault-end*)
with *Fault exec-bdy bdy While.hyps*
have $\Gamma\vdash_p\langle bdy1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash_p\langle bdy2,Normal\ s\rangle =n\Rightarrow Fault$
f
by *auto*
with *s-in-b* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
next
case (*Normal s‴*)
with *inter-guards-execn-noFault* [*OF bdy exec-bdy*]
obtain $\Gamma\vdash_p\langle bdy1,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
$\Gamma\vdash_p\langle bdy2,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
by *auto*
moreover
from *Normal WhileTrue*
have $\Gamma\vdash_p\langle While\ b\ bdy1,Normal\ s'''\rangle =n\Rightarrow Fault\ f \vee$
$\Gamma\vdash_p\langle While\ b\ bdy2,Normal\ s'''\rangle =n\Rightarrow Fault\ f$
by *simp*
ultimately show *?thesis*
using *s-in-b* **by** (*fastforce intro*: *execn.intros*)
next

case ($Abrupt\ s'''$)
with *exec-w Fault-s''* **show** *?thesis* **by** (*fastforce dest*: *execn-Abrupt-end*)
next
case *Stuck*
with *exec-w Fault-s''* **show** *?thesis* **by** (*fastforce dest*: *execn-Stuck-end*)
qed
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.intros*)
qed (*simp-all*)
}
with *this* [*OF exec-c c*] *c2*
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Call*)
next
case (*DynCom f1*)
have ($DynCom\ f1 \cap_{gs} c2$) = *Some c* **by** *fact*
then obtain *f2* **where**
c2: *c2=DynCom f2* **and**
F-defined: $\forall s.\ ((f1\ s) \cap_{gs} (f2\ s)) \neq None$ **and**
c: $c=DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_{gs} (f2\ s)))$
by (*auto simp add*: *inter-guards-DynCom*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
with *c* **have** $\Gamma\vdash_p\langle DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_{gs} (f2\ s))),Normal\ s\rangle =n\Rightarrow Fault\ f$
by *simp*
then show *?case*
proof (*cases*)
assume *exec-F*: $\Gamma\vdash_p\langle the\ (f1\ s \cap_{gs} f2\ s),Normal\ s\rangle =n\Rightarrow Fault\ f$
from *F-defined* **obtain** *F* **where** ($f1\ s \cap_{gs} f2\ s$) = *Some F*
by *auto*
with *DynCom.hyps this exec-F c2*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
next
case (*Guard m g1 bdy1*)
have ($Guard\ m\ g1\ bdy1 \cap_{gs} c2$) = *Some c* **by** *fact*
then obtain *g2 bdy2 bdy* **where**
c2: *c2=Guard m g2 bdy2* **and**
bdy: ($bdy1 \cap_{gs} bdy2$) = *Some bdy* **and**
c: $c=Guard\ m\ (g1 \cap g2)\ bdy$
by (*auto simp add*: *inter-guards-Guard*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
with *c* **have** $\Gamma\vdash_p\langle Guard\ m\ (g1 \cap g2)\ bdy,Normal\ s\rangle =n\Rightarrow Fault\ f$
by *simp*
thus *?case*
proof (*cases*)
assume *f-m*: *Fault f* = *Fault m*

assume $s \notin g1 \cap g2$
hence $s \notin g1 \lor s \notin g2$
by *blast*
with *c2 f-m* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
assume $s \in g1 \cap g2$
moreover
assume $\Gamma\vdash_p\langle bdy,Normal\ s\rangle =n\Rightarrow Fault\ f$
with *bdy* **have** $\Gamma\vdash_p\langle bdy1,Normal\ s\rangle =n\Rightarrow Fault\ f \lor \Gamma\vdash_p\langle bdy2,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*rule Guard.hyps*)
ultimately show *?thesis*
using *c2*
by (*auto intro*: *execn.intros*)
qed
next
case *Throw* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Throw*)
next
case (*Catch a1 a2*)
have $(Catch\ a1\ a2 \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *b1 b2 d1 d2* **where**
c2: $c2=Catch\ b1\ b2$ **and**
d1: $(a1 \cap_{gs} b1) = Some\ d1$ **and** *d2*: $(a2 \cap_{gs} b2) = Some\ d2$ **and**
c: $c=Catch\ d1\ d2$
by (*auto simp add*: *inter-guards-Catch*)
have $\Gamma\vdash_p\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
with *c* **have** $\Gamma\vdash_p\langle Catch\ d1\ d2,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *simp*
thus *?case*
proof (*cases*)
fix s'
assume $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
from *inter-guards-execn-noFault* [*OF d1 this*] **obtain**
exec-a1: $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$ **and**
exec-b1: $\Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
by *simp*
moreover assume $\Gamma\vdash_p\langle d2,Normal\ s'\rangle =n\Rightarrow Fault\ f$
with *d2*
have $\Gamma\vdash_p\langle a2,Normal\ s'\rangle =n\Rightarrow Fault\ f \lor \Gamma\vdash_p\langle b2,Normal\ s'\rangle =n\Rightarrow Fault\ f$
by (*auto dest*: *Catch.hyps*)
ultimately show *?thesis*
using *c2* **by** (*fastforce intro*: *execn.intros*)
next
assume $\Gamma\vdash_p\langle d1,Normal\ s\rangle =n\Rightarrow Fault\ f$
with *d1* **have** $\Gamma\vdash_p\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f \lor \Gamma\vdash_p\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*auto dest*: *Catch.hyps*)
with *c2* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)

qed
next
case (*Await b bdy1 e*)
have (*Await b bdy1 e* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
then obtain *bdy2 bdy* **where**
c2: *c2=Await b bdy2 e* **and**
bdy: (*bdy1* $\cap_g$ *bdy2*) = *Some bdy* **and**
c: *c=Await b bdy e*
by (*auto simp add*: *inter-guards-Await*)
have *exec-c*: $\Gamma\vdash_p\langle c,Normal\ s\rangle$ =*n*⇒ *Fault f* **by** *fact*
{
fix *s t n w*
assume *exec-w*: $\Gamma\vdash_p\langle w,Normal\ s\rangle$ =*n*⇒ *t*
assume *w*: *w=Await b bdy e*
assume *Fault*: *t=Fault f*
from *exec-w w Fault*
have $\Gamma\vdash_p\langle Await\ b\ bdy1\ e,Normal\ s\rangle$ =*n*⇒ *Fault f*∨
$\Gamma\vdash_p\langle Await\ b\ bdy2\ e,Normal\ s\rangle$ =*n*⇒ *Fault f*
using *SemanticCon.execn-Normal-elim-cases*(*11*) *bdy execn.AwaitTrue inter-guards-execn-Fault xstate.distinct*(*3*)
by (*metis*)

}
with *this* [*OF exec-c c*] *c2*
show *?case*
by *auto*
qed

lemma *inter-guards-execn-Fault*:
assumes *c*: (*c1* $\cap_{gs}$ *c2*) = *Some c*
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle$ =*n*⇒ *Fault f*
shows $\Gamma\vdash_p\langle c1,s\rangle$ =*n*⇒ *Fault f* ∨ $\Gamma\vdash_p\langle c2,s\rangle$ =*n*⇒ *Fault f*
proof (*cases s*)
case (*Fault f*)
with *exec-c* **show** *?thesis*
by (*auto dest*: *execn-Fault-end*)
next
case (*Abrupt s'*)
with *exec-c* **show** *?thesis*
by (*fastforce dest*: *execn-Abrupt-end*)
next
case *Stuck*
with *exec-c* **show** *?thesis*
by (*fastforce dest*: *execn-Stuck-end*)
next
case (*Normal s'*)
with *exec-c inter-guards-execn-Normal-Fault* [*OF c*]

show *?thesis*
by *blast*
qed

lemma *inter-guards-exec-Fault*:
assumes *c*: $(c1 \cap_{gs} c2) = Some\ c$
assumes *exec-c*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow Fault\ f$
shows $\Gamma\vdash_p\langle c1,s\rangle \Rightarrow Fault\ f \vee \Gamma\vdash_p\langle c2,s\rangle \Rightarrow Fault\ f$
proof −
from *exec-c* **obtain** n **where** $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow Fault\ f$
by (*auto simp add*: *exec-iff-execn*)
from *c this*
have $\Gamma\vdash_p\langle c1,s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash_p\langle c2,s\rangle =n\Rightarrow Fault\ f$
by (*rule inter-guards-execn-Fault*)
thus *?thesis*
by (*auto intro*: *execn-to-exec*)
qed

6.9 Restriction of Procedure Environment

lemma *restrict-SomeD*: $(m|_A)\ x = Some\ y \Longrightarrow m\ x = Some\ y$
by (*auto simp add*: *restrict-map-def split*: *if-split-asm*)

lemma *restrict-dom-same* [*simp*]: $m|_{dom\ m} = m$
apply (*rule ext*)
apply (*clarsimp simp add*: *restrict-map-def*)
apply (*simp only*: *not-None-eq* [*symmetric*])
apply *rule*
apply (*drule sym*)
apply *blast*
done

lemma *restrict-in-dom*: $x \in A \Longrightarrow (m|_A)\ x = m\ x$
by (*auto simp add*: *restrict-map-def*)

lemma *restrict-eq*: $(\Gamma|_A)_{\neg a} = (\Gamma_{\neg a})|_A$
unfolding *no-await-body-def*
apply *rule*
apply (*split option.split*)
apply *auto*
apply (*auto simp add*:*restrict-map-def*)
by (*meson option.distinct*(*1*))

lemma *exec-restrict-to-exec*:
assumes *exec-restrict*: $\Gamma|_A\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *notStuck*: $t\neq Stuck$
shows $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$

using *exec-restrict notStuck*
proof (*induct*)
  case (*AwaitTrue s b* Γ_p *ca t*)
  have $\Gamma_{\neg a}|_A = \Gamma_p$
    by (*simp add*: *AwaitTrue.hyps*(*2*) *restrict-eq*)
  hence $\Gamma_{\neg a}\vdash \langle ca, Normal\ s\rangle \Rightarrow t$
    using *AwaitTrue.hyps*(*3*) *AwaitTrue.prems exec-restrict-to-exec* **by** *blast*
  thus *?case*
    by (*simp add*: *AwaitTrue.hyps*(*1*) *exec.AwaitTrue*)
qed (*auto intro*: *exec.intros dest*: *restrict-SomeD Stuck-end*)

lemma *execn-restrict-to-execn*:
  assumes *exec-restrict*: $\Gamma|_A\vdash_p\langle c,s\rangle =n\Rightarrow t$
  assumes *notStuck*: $t\neq Stuck$
  shows $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
using *exec-restrict notStuck*
proof (*induct*)
 case (*AwaitTrue s b* Γ_p *ca n t*)
  have $\Gamma_{\neg a}|_A = \Gamma_p$
    by (*simp add*: *AwaitTrue.hyps*(*2*) *restrict-eq*)
  hence $\Gamma_{\neg a}\vdash \langle ca, Normal\ s\rangle =n\Rightarrow t$
    using *AwaitTrue.hyps*(*3*) *AwaitTrue.prems execn-restrict-to-execn* **by** *blast*
  thus *?case*
    by (*simp add*: *AwaitTrue.hyps*(*1*) *execn.AwaitTrue*)
qed(*auto intro*: *execn.intros dest*: *restrict-SomeD execn-Stuck-end*)

lemma *restrict-NoneD*: $m\ x = None \Longrightarrow (m|_A)\ x = None$
  by (*auto simp add*: *restrict-map-def split*: *if-split-asm*)

lemma *execn-to-execn-restrict*:
  assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
  shows $\exists t'.\ \Gamma|_P\vdash_p\langle c,s\rangle =n\Rightarrow t' \wedge (t=Stuck \longrightarrow t'=Stuck) \wedge$
        $(\forall f.\ t=Fault\ f \longrightarrow t'\in\{Fault\ f, Stuck\}) \wedge (t'\neq Stuck \longrightarrow t'=t)$
using *execn*
proof (*induct*)
  case *Skip* **show** *?case* **by** (*blast intro*: *execn.Skip*)
next
  case *Guard* **thus** *?case* **by** (*auto intro*: *execn.Guard*)
next
  case *GuardFault* **thus** *?case* **by** (*auto intro*: *execn.GuardFault*)
next
  case *FaultProp* **thus** *?case* **by** (*auto intro*: *execn.FaultProp*)
next
  case *Basic* **thus** *?case* **by** (*auto intro*: *execn.Basic*)
next
  case *Spec* **thus** *?case* **by** (*auto intro*: *execn.Spec*)
next

case *SpecStuck* **thus** *?case* **by** (*auto intro*: *execn.SpecStuck*)
next
case *Seq* **thus** *?case* **by** (*metis insertCI execn.Seq StuckProp*)
next
case *CondTrue* **thus** *?case* **by** (*auto intro*: *execn.CondTrue*)
next
case *CondFalse* **thus** *?case* **by** (*auto intro*: *execn.CondFalse*)
next
case *WhileTrue* **thus** *?case* **by** (*metis insertCI execn.WhileTrue StuckProp*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.WhileFalse*)
next
case (*Call p bdy n s s'*)
have Γ *p* = *Some bdy* **by** *fact*
show *?case*
proof (*cases* $p \in P$)
case *True*
with *Call* **have** $(\Gamma|_P)$ *p* = *Some bdy*
by (*simp*)
with *Call* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *False*
hence $(\Gamma|_P)$ *p* = *None* **by** *simp*
thus *?thesis*
by (*auto intro*: *execn.CallUndefined*)
qed
next
case (*CallUndefined p n s*)
have Γ *p* = *None* **by** *fact*
hence $(\Gamma|_P)$ *p* = *None* **by** (*rule restrict-NoneD*)
thus *?case* **by** (*auto intro*: *execn.CallUndefined*)
next
case *StuckProp* **thus** *?case* **by** (*auto intro*: *execn.StuckProp*)
next
case *DynCom* **thus** *?case* **by** (*auto intro*: *execn.DynCom*)
next
case *Throw* **thus** *?case* **by** (*auto intro*: *execn.Throw*)
next
case *AbruptProp* **thus** *?case* **by** (*auto intro*: *execn.AbruptProp*)
next
case (*CatchMatch c1 s n s' c2 s''*)
from *CatchMatch.hyps*
obtain t' t'' **where**
exec-res-c1: $\Gamma|_P \vdash_p \langle c1, Normal\ s\rangle =n\Rightarrow t'$ **and**
t'-notStuck: $t' \neq Stuck \longrightarrow t' = Abrupt\ s'$ **and**
exec-res-c2: $\Gamma|_P \vdash_p \langle c2, Normal\ s'\rangle =n\Rightarrow t''$ **and**
s''-Stuck: $s'' = Stuck \longrightarrow t'' = Stuck$ **and**
s''-Fault: $\forall f.\ s'' = Fault\ f \longrightarrow t'' \in \{Fault\ f,\ Stuck\}$ **and**

t''-*notStuck*: $t'' \neq$ *Stuck* $\longrightarrow t'' = s''$
by *auto*
show *?case*
proof (*cases* t'=*Stuck*)
case *True*
with *exec-res-c1*
have $\Gamma|_P\vdash_p\langle$*Catch c1 c2*,*Normal s*$\rangle$ =$n\Rightarrow$ *Stuck*
by (*auto intro*: *execn.CatchMiss*)
thus *?thesis*
by *auto*
next
case *False*
with t'-*notStuck* **have** t'= *Abrupt* s'
by *simp*
with *exec-res-c1 exec-res-c2*
have $\Gamma|_P\vdash_p\langle$*Catch c1 c2*,*Normal s*$\rangle$ =$n\Rightarrow$ t''
by (*auto intro*: *execn.CatchMatch*)
with s''-*Stuck* s''-*Fault* t''-*notStuck*
show *?thesis*
by *blast*
qed
next
case (*CatchMiss c1 s n w c2*)
have *exec-c1*: $\Gamma\vdash_p\langle$*c1*,*Normal s*$\rangle$ =$n\Rightarrow$ *w* **by** *fact*
from *CatchMiss.hyps* **obtain** w' **where**
exec-c1$'$: $\Gamma|_P\vdash_p\langle$*c1*,*Normal s*$\rangle$ =$n\Rightarrow$ w' **and**
w-Stuck: w = *Stuck* $\longrightarrow w'$ = *Stuck* **and**
w-Fault: $\forall f.\ w$ = *Fault f* $\longrightarrow w' \in \{$*Fault f*, *Stuck*$\}$ **and**
w'-*noStuck*: $w' \neq$ *Stuck* $\longrightarrow w' = w$
by *auto*
have *noAbr-w*: $\neg$ *isAbr w* **by** *fact*
show *?case*
proof (*cases* w')
case (*Normal* s')
with w'-*noStuck* **have** w'=w
by *simp*
with *exec-c1*$'$ *Normal w-Stuck w-Fault* w'-*noStuck*
show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
next
case (*Abrupt* s')
with w'-*noStuck* **have** w'=w
by *simp*
with *noAbr-w Abrupt* **show** *?thesis* **by** *simp*
next
case (*Fault f*)
with w'-*noStuck* **have** w'=w
by *simp*
with *exec-c1*$'$ *Fault w-Stuck w-Fault* w'-*noStuck*

show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
next
case *Stuck*
with *exec-c1′ w-Stuck w-Fault w′-noStuck*
show *?thesis*
by (*fastforce intro*: *execn.CatchMiss*)
qed
next
case ($AwaitTrue\ s\ b\ \Gamma_p\ c\ n\ t$)
have $\Gamma_{\neg a}|_P = (\Gamma|_P)_{\neg a}$
by (*simp add*: *AwaitTrue.hyps*(*2*) *restrict-eq*)
thus *?case* **using** *execn-to-execn-restrict* **by** (*metis* (*full-types*) *AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *AwaitTrue.hyps*(*3*) *execn.AwaitTrue*)
next
case (*AwaitFalse s b*) **thus** *?case* **by** (*fastforce intro*: *execn.AwaitFalse*)
qed

lemma *exec-to-exec-restrict*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
shows $\exists t'.\ \Gamma|_P\vdash_p\langle c,s\rangle \Rightarrow t' \wedge (t{=}Stuck \longrightarrow t'{=}Stuck) \wedge$
$(\forall f.\ t{=}Fault\ f \longrightarrow t'{\in}\{Fault\ f,Stuck\}) \wedge (t'{\neq}Stuck \longrightarrow t'{=}t)$
proof −
from *exec* **obtain** *n* **where**
execn-strip: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
by (*auto simp add*: *exec-iff-execn*)
from *execn-to-execn-restrict* [**where** *P*=*P*,*OF this*]
obtain *t′* **where**
$\Gamma|_P\vdash_p\langle c,s\rangle =n\Rightarrow t'$
$t{=}Stuck \longrightarrow t'{=}Stuck$ $\forall f.\ t{=}Fault\ f \longrightarrow t'{\in}\{Fault\ f,Stuck\}$ $t'{\neq}Stuck \longrightarrow t'{=}t$
by *blast*
thus *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

lemma *notStuck-GuardD*:
$[\![\Gamma\vdash_p\langle Guard\ m\ g\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in g]\!] \Longrightarrow \Gamma\vdash_p\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Guard*)

lemma *notStuck-SeqD1*:
$[\![\Gamma\vdash_p\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow \Gamma\vdash_p\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-SeqD2*:
$[\![\Gamma\vdash_p\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p\langle c1,Normal\ s\rangle \Rightarrow s']\!] \Longrightarrow \Gamma\vdash_p\langle c2,s'\rangle$
$\Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-SeqD*:
$[\![\Gamma\vdash_p\langle Seq\ c1\ c2, Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow$
$\Gamma\vdash_p\langle c1, Normal\ s\rangle \Rightarrow\notin\{Stuck\} \wedge (\forall s'.\ \Gamma\vdash_p\langle c1, Normal\ s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p\langle c2, s'\rangle$
$\Rightarrow\notin\{Stuck\})$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

lemma *notStuck-CondTrueD*:
$[\![\Gamma\vdash_p\langle Cond\ b\ c1\ c2, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b]\!] \Longrightarrow \Gamma\vdash_p\langle c1, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CondTrue*)

lemma *notStuck-CondFalseD*:
$[\![\Gamma\vdash_p\langle Cond\ b\ c1\ c2, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \notin b]\!] \Longrightarrow \Gamma\vdash_p\langle c2, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CondFalse*)

lemma *notStuck-WhileTrueD1*:
$[\![\Gamma\vdash_p\langle While\ b\ c, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b]\!]$
$\Longrightarrow \Gamma\vdash_p\langle c, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.WhileTrue*)

lemma *notStuck-WhileTrueD2*:
$[\![\Gamma\vdash_p\langle While\ b\ c, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\vdash_p\langle c, Normal\ s\rangle \Rightarrow s';\ s \in b]\!]$
$\Longrightarrow \Gamma\vdash_p\langle While\ b\ c, s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.WhileTrue*)

lemma *notStuck-AwaitTrueD1*:
$[\![\Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b]\!]$
$\Longrightarrow \exists\Gamma 1.\ \Gamma 1\vdash\langle c, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*meson Semantic.noStuckI' SemanticCon.noStuck-def' exec.AwaitTrue*)

lemma *notStuck-AwaitTrueD2*:
$[\![\Gamma 1\vdash\langle c, Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b;\ \Gamma 1{=}\Gamma_{\neg a}]\!]$
$\Longrightarrow \Gamma\vdash_p\langle Await\ b\ c\ e, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
unfolding *Semantic.final-notin-def final-notin-def*
by (*meson SemanticCon.exec-Normal-elim-cases*(*11*))

lemma *notStuck-CallD*:
$[\![\Gamma\vdash_p\langle Call\ p\ , Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ \Gamma\ p = Some\ bdy]\!]$
$\Longrightarrow \Gamma\vdash_p\langle bdy, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Call*)

lemma *notStuck-CallDefinedD*:
$[\![\Gamma\vdash_p\langle Call\ p, Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!]$
$\Longrightarrow \Gamma\ p \neq None$
by (*cases* Γ *p*)
(*auto simp add*: *final-notin-def dest*: *exec.CallUndefined*)

lemma *notStuck-DynComD*:
$[\![\Gamma\vdash_p\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!]$
$\Longrightarrow \Gamma\vdash_p\langle(c\ s),Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.DynCom*)

lemma *notStuck-CatchD1*:
$[\![\Gamma\vdash_p\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow \Gamma\vdash_p\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CatchMatch exec.CatchMiss*)

lemma *notStuck-CatchD2*:
$[\![\Gamma\vdash_p\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}; \Gamma\vdash_p\langle c1,Normal\ s\rangle \Rightarrow Abrupt\ s']\!]$
$\Longrightarrow \Gamma\vdash_p\langle c2,Normal\ s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.CatchMatch*)

6.10 Miscellaneous

lemma *no-guards-bdy*:$\Gamma 1=\Gamma_{\neg a} \Longrightarrow$
$\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
$\Longrightarrow \forall p \in dom\ \Gamma 1.\ Language.noguards\ (the\ (\Gamma 1\ p))$
proof
fix *p*
assume *a1*:$\Gamma 1 = \Gamma_{\neg a}$
assume *a2*:$\forall p\in dom\ \Gamma.\ LanguageCon.noguards\ (the\ (\Gamma\ p))$
assume *a3*:$p \in dom\ \Gamma 1$
with *a1 a2* **obtain** *t* **where** *t*:$\Gamma\ p = Some\ t$
by (*meson domD in-gamma-in-noawait-gamma*)
with *a3* **obtain** *s* **where** *s*:$\Gamma 1\ p = Some\ s$ **by** *blast*
with *t s a1* **have** *noaw-t*:*noawaits t* **by** (*meson no-await-some-no-await*)
with *a1 a3 s t lam1-seq* **have** *s=sequential t* **by** *fastforce*
moreover have *LanguageCon.noguards t*
using *a2 t* **by** *force*
ultimately have *Language.noguards s*
using *noaw-t noawaits-noguards-seq* **by** *blast*
then show *Language.noguards* (*the* ($\Gamma 1$ *p*))**by** (*simp add*: *s*)
qed

lemma *execn-noguards-no-Fault*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
assumes *noguards-c*: *noguards c*
assumes *noguards-*Γ: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
assumes *s-no-Fault*: $\neg$ *isFault s*
shows $\neg$ *isFault t*
using *execn noguards-c s-no-Fault*
proof (*induct*)
case (*Call p bdy n s t*) **with** *noguards-*Γ **show** *?case*
apply −
apply (*drule bspec* [**where** *x=p*])
apply *auto*
done

```
  next
    case (AwaitTrue s b Γ1 c n t)
      with Semantic.execn-noguards-no-Fault no-guards-bdy
      have s1:∀ p ∈ dom Γ1. Language.noguards (the (Γ1 p)) using  noguards-Γ
      proof −
        have ∀ a. a ∉ dom Γ1 ∨ Language.noguards (the (Γ1 a))
          by (metis (no-types) AwaitTrue.hyps(2) no-guards-bdy noguards-Γ)
        then show ?thesis
          by metis
      qed
      have Language.noguards c
        using AwaitTrue.prems(1) LanguageCon.noguards.simps(12) by blast
      hence ¬ Semantic.isFault t
      by (meson AwaitTrue.hyps(3) Semantic.isFault-simps(1) s1 execn-noguards-no-Fault)

      thus ?case
        using SemanticCon.not-isFault-iff by force
  qed (auto)

lemma exec-noguards-no-Fault:
 assumes exec: Γ⊢p⟨c,s⟩ ⇒ t
 assumes noguards-c: noguards c
 assumes noguards-Γ: ∀ p ∈ dom Γ. noguards (the (Γ p))
 assumes s-no-Fault: ¬ isFault s
 shows ¬ isFault t
  using exec noguards-c s-no-Fault
  proof (induct)
    case (Call p bdy s t) with noguards-Γ show ?case
      apply −
      apply (drule bspec [where x=p])
      apply auto
      done
  next
   case (AwaitTrue) thus ?case
     by (meson Semantic.exec-to-execn SemanticCon.execn-noguards-no-Fault ex-
ecn.AwaitTrue noguards-Γ)
  qed auto


lemma no-throws-bdy:Γ1=Γ¬a ⟹ ∀ p ∈ dom Γ. nothrows (the (Γ p))
                    ⟹ ∀ p ∈ dom Γ1. Language.nothrows (the (Γ1 p))
proof
  fix p
  assume a1:Γ1 = Γ¬a
  assume a2:∀ p∈dom Γ. LanguageCon.nothrows (the (Γ p))
  assume a3:p ∈ dom Γ1
  with a1 a2 obtain t where t:Γ p = Some t
    by (meson domD in-gamma-in-noawait-gamma)
  with a3 obtain s where s:Γ1 p = Some s by blast
```

with *t s a1* **have** *noaw-t*:*noawaits t* **by** (*meson no-await-some-no-await*)
with *a1 a3 s t lam1-seq* **have** *s*=*sequential t* **by** *fastforce*
moreover have *LanguageCon.nothrows t*
using *a2 t* **by** *force*
ultimately have *Language.nothrows s*
using *noaw-t noawaits-nothrows-seq* **by** *blast*
then show *Language.nothrows* (*the* (Γ*1 p*))**by** (*simp add*: *s*)
qed

lemma *execn-nothrows-no-Abrupt*:
assumes *execn*: $\Gamma\vdash_p\langle c,s\rangle =n\Rightarrow t$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: $\forall\, p \in$ *dom* Γ. *nothrows* (*the* (Γ *p*))
assumes *s-no-Abrupt*: $\neg$(*isAbr s*)
shows $\neg$(*isAbr t*)
using *execn nothrows-c s-no-Abrupt*
proof (*induct*)
case (*Call p bdy n s t*) **with** *nothrows-*Γ **show** *?case*
apply −
apply (*drule bspec* [**where** *x*=*p*])
apply *auto*
done
next
case (*AwaitTrue s b* Γ*1 c n t*)
with *Semantic.execn-noguards-no-Fault no-throws-bdy*
have *s*:$\forall\, p \in$ *dom* Γ*1*. *Language.nothrows* (*the* (Γ*1 p*)) **using** *nothrows-*Γ
proof −
have $\forall\, a.\ a \notin$ *dom* Γ*1* $\vee$ *Language.nothrows* (*the* (Γ*1 a*))
by (*simp add*: *AwaitTrue.hyps*(*2*) *no-throws-bdy nothrows-*Γ)
then show *?thesis*
by *metis*
qed
have *Language.nothrows c*
using *AwaitTrue.prems*(*1*) *LanguageCon.nothrows.simps*(*12*) **by** *blast*
hence $\neg$ *Semantic.isAbr t*
by (*meson AwaitTrue.hyps*(*3*) *Semantic.execn-to-exec Semantic.isAbr-simps*(*1*) *s exec-nothrows-no-Abrupt*)
thus *?case* **using** *Semantic.isAbr-def SemanticCon.isAbrE* **by** *fastforce*
qed (*auto*)

lemma *exec-nothrows-no-Abrupt*:
assumes *exec*: $\Gamma\vdash_p\langle c,s\rangle \Rightarrow t$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: $\forall\, p \in$ *dom* Γ. *nothrows* (*the* (Γ *p*))
assumes *s-no-Abrupt*: $\neg$(*isAbr s*)
shows $\neg$(*isAbr t*)
using *exec nothrows-c s-no-Abrupt*
proof (*induct*)
case (*Call p bdy s t*) **with** *nothrows-*Γ **show** *?case*

apply −
apply (*drule bspec* [**where** $x=p$])
apply *auto*
done
next
case (*AwaitTrue*) **thus** *?case*
by (*meson Semantic.exec-to-execn execn-nothrows-no-Abrupt execn.AwaitTrue nothrows-*Γ)
qed (*auto*)

end

7 Terminating Programs

theory *TerminationCon* **imports** *SemanticCon EmbSimpl/Termination* **begin**

7.1 Inductive Characterisation: $\Gamma\vdash c\downarrow s$

inductive *terminates*::$('s,'p,'f,'e)$ *body* ⇒ $('s,'p,'f,'e)$ *com* ⇒ $('s,'f)$ *xstate* ⇒ *bool*
$(\text{-}\vdash_p\text{-} \downarrow \text{-}\ [60,20,60]\ 89)$
for $\Gamma::('s,'p,'f,'e)$ *body*
where
Skip: $\Gamma\vdash_p Skip\ \downarrow(Normal\ s)$

| *Basic*: $\Gamma\vdash_p Basic\ f\ e\ \downarrow(Normal\ s)$

| *Spec*: $\Gamma\vdash_p Spec\ r\ e\ \downarrow(Normal\ s)$

| *Guard*: $[\![s\in g;\ \Gamma\vdash_p c\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Guard\ f\ g\ c\downarrow(Normal\ s)$

| *GuardFault*: $s\notin g$
$\Longrightarrow$
$\Gamma\vdash_p Guard\ f\ g\ c\downarrow(Normal\ s)$

| *Fault* [*intro*,*simp*]: $\Gamma\vdash_p c\downarrow Fault\ f$

| *Seq*: $[\![\Gamma\vdash_p c_1\downarrow Normal\ s;\ \forall s'.\ \Gamma\vdash_p\langle c_1,Normal\ s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p c_2\downarrow s']\!]$
$\Longrightarrow$
$\Gamma\vdash_p Seq\ c_1\ \ c_2\downarrow(Normal\ s)$

| *CondTrue*: $[\![s \in b;\ \Gamma\vdash_p c_1\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Cond\ b\ c_1\ \ c_2\downarrow(Normal\ s)$

| *CondFalse*: $[\![s \notin b;\ \Gamma\vdash_p c_2\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Cond\ b\ c_1\ \ c_2\downarrow(Normal\ s)$

| *WhileTrue*: $[\![s \in b;\ \Gamma\vdash_p c\downarrow(Normal\ s);$
$\forall s'.\ \Gamma\vdash_p\langle c, Normal\ s\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p While\ b\ c\downarrow s']\!]$
$\Longrightarrow$
$\Gamma\vdash_p While\ b\ c\downarrow(Normal\ s)$

| *AwaitTrue*: $[\![s \in b;$
$\Gamma_p{=}\Gamma_{\neg a}\ ;\ \Gamma_p\vdash\ c\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Await\ b\ c\ e\downarrow(Normal\ s)$

| *AwaitFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Await\ b\ c\ e\downarrow(Normal\ s)$

| *WhileFalse*: $[\![s \notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash_p While\ b\ c\downarrow(Normal\ s)$

| *Call*: $[\![\Gamma\ p{=}Some\ bdy;\Gamma\vdash_p bdy\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Call\ p\downarrow(Normal\ s)$

| *CallUndefined*: $[\![\Gamma\ p\ =\ None]\!]$
$\Longrightarrow$
$\Gamma\vdash_p Call\ p\downarrow(Normal\ s)$

| *Stuck* [*intro*,*simp*]: $\Gamma\vdash_p c\downarrow Stuck$

| *DynCom*: $[\![\Gamma\vdash_p(c\ s)\downarrow(Normal\ s)]\!]$
$\Longrightarrow$
$\Gamma\vdash_p DynCom\ c\downarrow(Normal\ s)$

| *Throw*: $\Gamma\vdash_p Throw\downarrow(Normal\ s)$

| *Abrupt* [*intro*,*simp*]: $\Gamma\vdash_p c\downarrow Abrupt\ s$

| *Catch*: $[\![\Gamma\vdash_p c_1\downarrow Normal\ s;$
$\forall s'.\ \Gamma\vdash_p\langle c_1, Normal\ s\ \rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash_p c_2\downarrow Normal\ s']\!]$
$\Longrightarrow$
$\Gamma\vdash_p Catch\ c_1\ \ c_2\downarrow Normal\ s$

inductive-cases *terminates-elim-cases* [*cases set*]:
$\Gamma\vdash_p Skip\ \downarrow\ s$
$\Gamma\vdash_p Guard\ f\ g\ c\ \downarrow\ s$

$\Gamma\vdash_p Basic\ f\ e \downarrow s$
$\Gamma\vdash_p Spec\ r\ e \downarrow s$
$\Gamma\vdash_p Seq\ c1\ c2 \downarrow s$
$\Gamma\vdash_p Cond\ b\ c1\ c2 \downarrow s$
$\Gamma\vdash_p While\ b\ c \downarrow s$
$\Gamma\vdash_p Call\ p \downarrow s$
$\Gamma\vdash_p DynCom\ c \downarrow s$
$\Gamma\vdash_p Throw \downarrow s$
$\Gamma\vdash_p Catch\ c1\ c2 \downarrow s$
$\Gamma\vdash_p Await\ b\ c\ e \downarrow s$

inductive-cases *terminates-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_p Skip \downarrow Normal\ s$
$\Gamma\vdash_p Guard\ f\ g\ c \downarrow Normal\ s$
$\Gamma\vdash_p Basic\ f\ e \downarrow Normal\ s$
$\Gamma\vdash_p Spec\ r\ e \downarrow Normal\ s$
$\Gamma\vdash_p Seq\ c1\ c2 \downarrow Normal\ s$
$\Gamma\vdash_p Cond\ b\ c1\ c2 \downarrow Normal\ s$
$\Gamma\vdash_p While\ b\ c \downarrow Normal\ s$
$\Gamma\vdash_p Call\ p \downarrow Normal\ s$
$\Gamma\vdash_p DynCom\ c \downarrow Normal\ s$
$\Gamma\vdash_p Throw \downarrow Normal\ s$
$\Gamma\vdash_p Catch\ c1\ c2 \downarrow Normal\ s$
$\Gamma\vdash_p Await\ b\ c\ e \downarrow Normal\ s$

lemma *terminates-Skip'*: $\Gamma\vdash_p Skip \downarrow s$
by (*cases s*) (*auto intro*: *terminates.intros*)

lemma *terminates-Call-body*:
$\Gamma\ p=Some\ bdy \Longrightarrow \Gamma\vdash_p Call\ \ p \downarrow s = \Gamma\vdash_p (the\ (\Gamma\ p))\downarrow s$
by (*cases s*)
(*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*)

lemma *terminates-Normal-Call-body*:
$p \in dom\ \Gamma \Longrightarrow$
$\Gamma\vdash_p Call\ p \downarrow Normal\ s = \Gamma\vdash_p (the\ (\Gamma\ p))\downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*)

lemma *terminates-implies-exec*:
assumes *terminates*: $\Gamma\vdash_p c\downarrow s$
shows $\exists\, t.\ \Gamma\vdash_p \langle c,s\rangle \Rightarrow t$
using *terminates*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*iprover intro*: *exec.intros*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *exec.intros*)
next
case (*Spec r e s*) **thus** *?case*

```
    by (cases ∃ t. (s,t)∈ r) (auto intro: exec.intros)
next
  case Guard thus ?case by (iprover intro: exec.intros)
next
  case GuardFault thus ?case by (iprover intro: exec.intros)
next
  case Fault thus ?case by (iprover intro: exec.intros)
next
  case Seq thus ?case by (iprover intro: exec-Seq')
next
  case CondTrue thus ?case by (iprover intro: exec.intros)
next
  case CondFalse thus ?case by (iprover intro: exec.intros)
next
  case WhileTrue thus ?case by (iprover intro: exec.intros)
next
  case WhileFalse thus ?case by (iprover intro: exec.intros)
next
  case (Call p bdy s)
  then obtain s' where
    Γ⊢p ⟨bdy,Normal s ⟩ ⇒ s'
    by iprover
  moreover have Γ p = Some bdy by fact
  ultimately show ?case
    by (cases s') (iprover intro: exec.intros)+
next
  case CallUndefined thus ?case by (iprover intro: exec.intros)
next
  case Stuck thus ?case by (iprover intro: exec.intros)
next
  case DynCom thus ?case by (iprover intro: exec.intros)
next
  case Throw thus ?case by (iprover intro: exec.intros)
next
  case Abrupt thus ?case by (iprover intro: exec.intros)
next
  case (Catch c1 s c2)
  then obtain s' where exec-c1: Γ⊢p ⟨c1,Normal s ⟩ ⇒ s'
    by iprover
  thus ?case
  proof (cases s')
    case (Normal s'')
    with exec-c1 show ?thesis by (auto intro!: exec.intros)
  next
    case (Abrupt s'')
    with exec-c1 Catch.hyps
    obtain t where Γ⊢p ⟨c2,Normal s'' ⟩ ⇒ t
      by auto
    with exec-c1 Abrupt show ?thesis by (auto intro: exec.intros)
```

next
case *Fault*
with *exec-c1* **show** *?thesis* **by** (*auto intro*!: *exec.CatchMiss*)
next
case *Stuck*
with *exec-c1* **show** *?thesis* **by** (*auto intro*!: *exec.CatchMiss*)
qed
next
case (*AwaitTrue s b* Γ_p *c*)
then obtain *t* **where** $\Gamma_p\vdash\langle c,Normal\ s\rangle \Rightarrow t$
using *terminates-implies-exec* **by** *fastforce*
then have $\Gamma_{\neg a}\vdash \langle c,Normal\ s\rangle \Rightarrow t$
using *AwaitTrue.hyps*(*2*) ‹$\Gamma_p\vdash \langle c,Normal\ s\rangle \Rightarrow t$› **by** *blast*
thus *?case*
by (*meson AwaitTrue.hyps*(*1*) *exec.AwaitTrue*)
next
case (*AwaitFalse s b*) **thus** *?case* **by** (*fastforce intro*: *exec.intros*(*13*))
qed

lemma *terminates-block*:
$[\![\Gamma\vdash_p bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash_p c\ s\ t \downarrow Normal\ (return\ s\ t)]\!]$
$\Longrightarrow \Gamma\vdash_p block\ init\ ei\ bdy\ return\ er\ c \downarrow Normal\ s$
apply (*unfold block-def*)
apply (*fastforce intro*: *terminates.intros elim*!: *exec-Normal-elim-cases*
dest!: *not-isAbrD*)
done

lemma *terminates-block-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash_p block\ init\ ei\ bdy\ return\ er\ c \downarrow Normal\ s$
assumes *e*: $[\![\Gamma\vdash_p bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash_p c\ s\ t \downarrow Normal\ (return$
$s\ t)$
$]\!] \Longrightarrow P$
shows *P*
proof −
have $\Gamma\vdash_p\langle Basic\ init\ ei,Normal\ s\rangle \Rightarrow Normal\ (init\ s)$
by (*auto intro*: *exec.intros*)
with *termi*
have $\Gamma\vdash_p bdy \downarrow Normal\ (init\ s)$
apply (*unfold block-def*)
apply (*elim terminates-Normal-elim-cases*)
by *simp*
moreover
{
fix *t*
assume *exec-bdy*: $\Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
have $\Gamma\vdash_p c\ s\ t \downarrow Normal\ (return\ s\ t)$
proof −

from *exec-bdy*
have $\Gamma\vdash_p\langle Catch\ (Seq\ (Basic\ init\ ei)\ bdy)$
$(Seq\ (Basic\ (return\ s)\ er)\ Throw),Normal\ s\rangle \Rightarrow Normal\ t$
by (*fastforce intro*: *exec.intros*)
with *termi* **have** $\Gamma\vdash_p DynCom\ (\lambda t.\ Seq\ (Basic\ (return\ s)\ er)\ (c\ s\ t)) \downarrow Normal$
t
apply (*unfold block-def*)
apply (*elim terminates-Normal-elim-cases*)
by *simp*
thus *?thesis*
apply (*elim terminates-Normal-elim-cases*)
apply (*auto intro*: *exec.intros*)
done
qed
}
ultimately show *P* **by** (*iprover intro*: *e*)
qed

lemma *terminates-call*:
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash_p c\ s\ t \downarrow Normal\ (return\ s\ t)]\!]$
$\Longrightarrow \Gamma\vdash_p call\ init\ ei\ p\ return\ er\ c \downarrow Normal\ s$
apply (*unfold call-def*)
apply (*rule terminates-block*)
apply (*iprover intro*: *terminates.intros*)
apply (*auto elim*: *exec-Normal-elim-cases*)
done

lemma *terminates-callUndefined*:
$[\![\Gamma\ p = None]\!]$
$\Longrightarrow \Gamma\vdash_p call\ init\ ei\ p\ return\ er\ result \downarrow Normal\ s$
apply (*unfold call-def*)
apply (*rule terminates-block*)
apply (*iprover intro*: *terminates.intros*)
apply (*auto elim*: *exec-Normal-elim-cases*)
done

lemma *terminates-call-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash_p call\ init\ ei\ p\ return\ er\ c \downarrow Normal\ s$
assumes *bdy*: $\bigwedge bdy.\ [\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash_p bdy \downarrow Normal\ (init\ s);$
$\forall t.\ \Gamma\vdash_p\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow \Gamma\vdash_p c\ s\ t \downarrow Normal\ (return\ s$
$t)]\!] \Longrightarrow P$
assumes *undef*: $[\![\Gamma\ p = None]\!] \Longrightarrow P$
shows *P*
apply (*cases* Γ *p*)
apply (*erule undef*)
using *termi*
apply (*unfold call-def*)

apply (*erule terminates-block-elim*)
apply (*erule terminates-Normal-elim-cases*)
apply *simp*
apply (*frule* (*1*) *bdy*)
apply (*fastforce intro*: *exec.intros*)
apply *assumption*
apply *simp*
done

lemma *terminates-dynCall*:
$[\![\Gamma\vdash_p call\ init\ ei\ (p\ s)\ return\ er\ c \downarrow Normal\ s]\!]$
$\Longrightarrow \Gamma\vdash_p dynCall\ init\ ei\ p\ return\ er\ c \downarrow Normal\ s$
  apply (*unfold dynCall-def*)
  apply (*auto intro*: *terminates.intros terminates-call*)
  done

lemma *terminates-dynCall-elim* [*cases set*, *consumes 1*]:
assumes *termi*: $\Gamma\vdash_p dynCall\ init\ ei\ p\ return\ er\ c \downarrow Normal\ s$
assumes $[\![\Gamma\vdash_p call\ init\ ei\ (p\ s)\ return\ er\ c \downarrow Normal\ s]\!] \Longrightarrow P$
shows *P*
using *termi*
apply (*unfold dynCall-def*)
apply (*elim terminates-Normal-elim-cases*)
apply *fact*
done

7.2 Lemmas about *LanguageCon.sequence*, *LanguageCon.flatten* and *LanguageCon.normalize*

lemma *terminates-sequence-app*:
  $\bigwedge s.\ [\![\Gamma\vdash_p sequence\ Seq\ xs \downarrow Normal\ s;$
    $\forall s'.\ \Gamma\vdash_p\langle sequence\ Seq\ xs,Normal\ s\ \rangle \Rightarrow s' \longrightarrow\ \Gamma\vdash_p sequence\ Seq\ ys \downarrow s']\!]$
$\Longrightarrow \Gamma\vdash_p sequence\ Seq\ (xs\ @\ ys) \downarrow Normal\ s$
proof (*induct xs*)
  case *Nil*
  thus *?case* **by** (*auto intro*: *exec.intros*)
next
  case (*Cons x xs*)
  have *termi-x-xs*: $\Gamma\vdash_p sequence\ Seq\ (x\ \#\ xs) \downarrow Normal\ s$ **by** *fact*
  have *termi-ys*: $\forall s'.\ \Gamma\vdash_p\langle sequence\ Seq\ (x\ \#\ xs),Normal\ s\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p sequence$ $Seq\ ys \downarrow s'$ **by** *fact*
  show *?case*
  proof (*cases xs*)
    case *Nil*
    with *termi-x-xs termi-ys* **show** *?thesis*
      by (*cases ys*) (*auto intro*: *terminates.intros*)
  next
    case *Cons*
    from *termi-x-xs Cons*

have $\Gamma\vdash_p x \downarrow$ *Normal s*
by (*auto elim*: *terminates-Normal-elim-cases*)
moreover
{
fix s'
assume *exec-x*: $\Gamma\vdash_p\langle x,$*Normal s* $\rangle \Rightarrow s'$
have $\Gamma\vdash_p$*sequence Seq* (*xs* @ *ys*) $\downarrow s'$
proof −
from *exec-x termi-x-xs Cons*
have *termi-xs*: $\Gamma\vdash_p$*sequence Seq xs* $\downarrow s'$
by (*auto elim*: *terminates-Normal-elim-cases*)
show *?thesis*
proof (*cases* s')
case (*Normal* s'')
with *exec-x termi-ys Cons*
have $\forall s'.$ $\Gamma\vdash_p\langle$*sequence Seq xs*,*Normal* s'' $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p$*sequence Seq ys*
$\downarrow s'$
by (*auto intro*: *exec.intros*)
from *Cons.hyps* [*OF termi-xs* [*simplified Normal*] *this*]
have $\Gamma\vdash_p$*sequence Seq* (*xs* @ *ys*) $\downarrow$ *Normal* s''**.**
with *Normal* **show** *?thesis* **by** *simp*
next
case *Abrupt* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
next
case *Fault* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
next
case *Stuck* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
qed
qed
}
ultimately show *?thesis*
using *Cons*
by (*auto intro*: *terminates.intros*)
qed
qed

lemma *terminates-sequence-appD*:
$\bigwedge s.$ $\Gamma\vdash_p$*sequence Seq* (*xs* @ *ys*) $\downarrow$ *Normal s*
$\Longrightarrow \Gamma\vdash_p$*sequence Seq xs* $\downarrow$ *Normal s* $\wedge$
($\forall s'.$ $\Gamma\vdash_p\langle$*sequence Seq xs*,*Normal s* $\rangle \Rightarrow s' \longrightarrow$ $\Gamma\vdash_p$*sequence Seq ys* $\downarrow s'$)
proof (*induct xs*)
case *Nil*
thus *?case*
by (*auto elim*: *terminates-Normal-elim-cases exec-Normal-elim-cases*
intro: *terminates.intros*)
next
case (*Cons x xs*)
have *termi-x-xs-ys*: $\Gamma\vdash_p$*sequence Seq* ((*x* # *xs*) @ *ys*) $\downarrow$ *Normal s* **by** *fact*
show *?case*

proof (*cases xs*)
case *Nil*
with *termi-x-xs-ys* **show** *?thesis*
by (*cases ys*)
(*auto elim*: *terminates-Normal-elim-cases exec-Normal-elim-cases*
intro: *terminates-Skip'*)
next
case *Cons*
with *termi-x-xs-ys*
obtain *termi-x*: $\Gamma\vdash_p x \downarrow \mathit{Normal}\ s$ **and**
termi-xs-ys: $\forall s'.\ \Gamma\vdash_p\langle x,\mathit{Normal}\ s\ \rangle \Rightarrow s' \longrightarrow\ \Gamma\vdash_p \mathit{sequence\ Seq}\ (\mathit{xs}@\mathit{ys})$
$\downarrow s'$
by (*auto elim*: *terminates-Normal-elim-cases*)

have $\Gamma\vdash_p \mathit{Seq}\ x\ (\mathit{sequence\ Seq\ xs}) \downarrow \mathit{Normal}\ s$
proof (*rule terminates.Seq* [*rule-format*])
show $\Gamma\vdash_p x \downarrow \mathit{Normal}\ s$ **by** (*rule termi-x*)
next
fix s'
assume *exec-x*: $\Gamma\vdash_p\langle x,\mathit{Normal}\ s\ \rangle \Rightarrow s'$
show $\Gamma\vdash_p \mathit{sequence\ Seq\ xs} \downarrow s'$
proof −
from *termi-xs-ys* [*rule-format*, *OF exec-x*]
have *termi-xs-ys'*: $\Gamma\vdash_p \mathit{sequence\ Seq}\ (\mathit{xs}@\mathit{ys}) \downarrow s'$ **.**
show *?thesis*
proof (*cases s'*)
case (*Normal s''*)
from *Cons.hyps* [*OF termi-xs-ys'* [*simplified Normal*]]
show *?thesis*
using *Normal* **by** *auto*
next
case *Abrupt* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
next
case *Fault* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
next
case *Stuck* **thus** *?thesis* **by** (*auto intro*: *terminates.intros*)
qed
qed
qed
moreover
{
fix s'
assume *exec-x-xs*: $\Gamma\vdash_p\langle \mathit{Seq}\ x\ (\mathit{sequence\ Seq\ xs}),\mathit{Normal}\ s\ \rangle \Rightarrow s'$
have $\Gamma\vdash_p \mathit{sequence\ Seq\ ys} \downarrow s'$
proof −
from *exec-x-xs* **obtain** *t* **where**
exec-x: $\Gamma\vdash_p\langle x,\mathit{Normal}\ s\ \rangle \Rightarrow t$ **and**
exec-xs: $\Gamma\vdash_p\langle \mathit{sequence\ Seq\ xs},t\ \rangle \Rightarrow s'$
by *cases*

```
        show ?thesis
        proof (cases t)
          case (Normal t')
          with exec-x termi-xs-ys have Γ⊢p sequence Seq (xs@ys) ↓ Normal t'
            by auto
          from Cons.hyps [OF this] exec-xs Normal
          show ?thesis
            by auto
        next
          case (Abrupt t')
          with exec-xs have s'=Abrupt t'
            by (auto dest: Abrupt-end)
          thus ?thesis by (auto intro: terminates.intros)
        next
          case (Fault f)
          with exec-xs have s'=Fault f
            by (auto dest: Fault-end)
          thus ?thesis by (auto intro: terminates.intros)
        next
          case Stuck
          with exec-xs have s'=Stuck
            by (auto dest: Stuck-end)
          thus ?thesis by (auto intro: terminates.intros)
        qed
      qed
    }
    ultimately show ?thesis
      using Cons
      by auto
  qed
qed

lemma terminates-sequence-appE [consumes 1]:
  ⟦Γ⊢p sequence Seq (xs @ ys) ↓ Normal s;
    ⟦Γ⊢p sequence Seq xs ↓ Normal s;
     ∀ s'. Γ⊢p ⟨sequence Seq xs,Normal s ⟩ ⇒ s' ⟶ Γ⊢p sequence Seq ys ↓ s'⟧ ⟹
P⟧
   ⟹ P
  by (auto dest: terminates-sequence-appD)

lemma terminates-to-terminates-sequence-flatten:
  assumes termi: Γ⊢p c↓s
  shows Γ⊢p sequence Seq (flatten c)↓s
using termi
by (induct)
   (auto intro: terminates.intros terminates-sequence-app
     exec-sequence-flatten-to-exec)

lemma terminates-to-terminates-normalize:
```

assumes *termi*: $\Gamma\vdash_p c\downarrow s$
shows $\Gamma\vdash_p$*normalize c*$\downarrow s$
using *termi*
proof *induct*
case *Seq*
thus *?case*
by (*fastforce intro*: *terminates.intros terminates-sequence-app*
terminates-to-terminates-sequence-flatten
dest: *exec-sequence-flatten-to-exec exec-normalize-to-exec*)
next
case *WhileTrue*
thus *?case*
by (*fastforce intro*: *terminates.intros terminates-sequence-app*
terminates-to-terminates-sequence-flatten
dest: *exec-sequence-flatten-to-exec exec-normalize-to-exec*)
next
case *Catch*
thus *?case*
by (*fastforce intro*: *terminates.intros terminates-sequence-app*
terminates-to-terminates-sequence-flatten
dest: *exec-sequence-flatten-to-exec exec-normalize-to-exec*)
next
case *AwaitTrue*
thus *?case*
using *terminates-to-terminates-normalize*
by (*simp add*: *terminates-to-terminates-normalize terminates.AwaitTrue*)
qed (*auto intro*: *terminates.intros*)

lemma *terminates-sequence-flatten-to-terminates*:
shows $\bigwedge s.\ \Gamma\vdash_p$*sequence Seq* (*flatten c*)$\downarrow s \Longrightarrow \Gamma\vdash_p c\downarrow s$
proof (*induct c*)
case (*Seq c1 c2*)
have $\Gamma\vdash_p$*sequence Seq* (*flatten* (*Seq c1 c2*)) $\downarrow$ *s* **by** *fact*
hence *termi-app*: $\Gamma\vdash_p$*sequence Seq* (*flatten c1* @ *flatten c2*) $\downarrow$ *s* **by** *simp*
show *?case*
proof (*cases s*)
case (*Normal s'*)
have $\Gamma\vdash_p$*Seq c1 c2* $\downarrow$ *Normal s'*
proof (*rule terminates.Seq* [*rule-format*])
from *termi-app* [*simplified Normal*]
have $\Gamma\vdash_p$*sequence Seq* (*flatten c1*) $\downarrow$ *Normal s'*
by (*cases rule*: *terminates-sequence-appE*)
with *Seq.hyps*
show $\Gamma\vdash_p$*c1* $\downarrow$ *Normal s'*
by *simp*
next
fix *s''*
assume $\Gamma\vdash_p\langle$*c1*,*Normal s'*$\rangle \Rightarrow$ *s''*
from *termi-app* [*simplified Normal*] *exec-to-exec-sequence-flatten* [*OF this*]

have $\Gamma\vdash_p$*sequence Seq* (*flatten c2*) $\downarrow$ *s''*
by (*cases rule*: *terminates-sequence-appE*) *auto*
with *Seq.hyps*
show $\Gamma\vdash_p$*c2* $\downarrow$ *s''*
by *simp*
qed
with *Normal* **show** *?thesis*
by *simp*
qed (*auto intro*: *terminates.intros*)
qed (*auto intro*: *terminates.intros*)

lemma *terminates-normalize-to-terminates*:
shows $\bigwedge s.\ \Gamma\vdash_p$*normalize c*$\downarrow$*s* $\Longrightarrow$ $\Gamma\vdash_p$*c*$\downarrow$*s*
proof (*induct c*)
case *Skip* **thus** *?case* **by** (*auto intro*: *terminates-Skip'*)
next
case *Basic* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
case *Spec* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
case (*Seq c1 c2*)
have $\Gamma\vdash_p$*normalize* (*Seq c1 c2*) $\downarrow$ *s* **by** *fact*
hence *termi-app*: $\Gamma\vdash_p$*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*)) $\downarrow$ *s*
by *simp*
show *?case*
proof (*cases s*)
case (*Normal s'*)
have $\Gamma\vdash_p$*Seq c1 c2* $\downarrow$ *Normal s'*
proof (*rule terminates.Seq* [*rule-format*])
from *termi-app* [*simplified Normal*]
have $\Gamma\vdash_p$*sequence Seq* (*flatten* (*normalize c1*)) $\downarrow$ *Normal s'*
by (*cases rule*: *terminates-sequence-appE*)
from *terminates-sequence-flatten-to-terminates* [*OF this*] *Seq.hyps*
show $\Gamma\vdash_p$*c1* $\downarrow$ *Normal s'*
by *simp*
next
fix *s''*
assume $\Gamma\vdash_p\langle$*c1*,*Normal s'* $\rangle \Rightarrow$ *s''*
from *exec-to-exec-normalize* [*OF this*]
have $\Gamma\vdash_p\langle$*normalize c1*,*Normal s'* $\rangle \Rightarrow$ *s''* **.**
from *termi-app* [*simplified Normal*] *exec-to-exec-sequence-flatten* [*OF this*]
have $\Gamma\vdash_p$*sequence Seq* (*flatten* (*normalize c2*)) $\downarrow$ *s''*
by (*cases rule*: *terminates-sequence-appE*) *auto*
from *terminates-sequence-flatten-to-terminates* [*OF this*] *Seq.hyps*
show $\Gamma\vdash_p$*c2* $\downarrow$ *s''*
by *simp*
qed
with *Normal* **show** *?thesis* **by** *simp*

qed (*auto intro*: *terminates.intros*)
next
case (*Cond b c1 c2*)
thus *?case*
by (*cases s*)
(*auto intro*: *terminates.intros elim*!: *terminates-Normal-elim-cases*)
next
case (*While b c*)
have $\Gamma\vdash_p$*normalize* (*While b c*) $\downarrow$ *s* **by** *fact*
hence *termi-norm-w*: $\Gamma\vdash_p$ *While b* (*normalize c*) $\downarrow$ *s* **by** *simp*
{
fix *t w*
assume *termi-w*: $\Gamma\vdash_p$ *w* $\downarrow$ *t*
have *w*=*While b* (*normalize c*) $\Longrightarrow$ $\Gamma\vdash_p$ *While b c* $\downarrow$ *t*
using *termi-w*
proof (*induct*)
case (*WhileTrue t' b' c'*)
from *WhileTrue* **obtain**
t'-b: $t' \in b$ **and**
termi-norm-c: $\Gamma\vdash_p$*normalize c* $\downarrow$ *Normal t'* **and**
termi-norm-w': $\forall s'.$ $\Gamma\vdash_p\langle$*normalize c*,*Normal t'* $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p$ *While b c* $\downarrow$
s'
by *auto*
from *While.hyps* [*OF termi-norm-c*]
have $\Gamma\vdash_p$*c* $\downarrow$ *Normal t'*.
moreover
from *termi-norm-w'*
have $\forall s'.$ $\Gamma\vdash_p\langle$*c*,*Normal t'* $\rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p$ *While b c* $\downarrow$ *s'*
by (*auto intro*: *exec-to-exec-normalize*)
ultimately show *?case*
using *t'-b*
by (*auto intro*: *terminates.intros*)
qed (*auto intro*: *terminates.intros*)
}
from *this* [*OF termi-norm-w*]
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** *simp*
next
case *DynCom* **thus** *?case*
by (*cases s*) (*auto intro*: *terminates.intros rangeI elim*: *terminates-Normal-elim-cases*)
next
case *Guard* **thus** *?case*
by (*cases s*) (*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *Throw* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
case *Catch*

thus *?case*
by (*cases s*)
(*auto dest*: *exec-to-exec-normalize elim*!: *terminates-Normal-elim-cases*
intro!: *terminates.Catch*)
next
case (*Await b c*) **thus** *?case*
by (*cases s*) (*auto intro*: *terminates-normalize-to-terminates terminates.AwaitTrue*
terminates.AwaitFalse rangeI elim: *terminates-Normal-elim-cases*)
qed

lemma *terminates-iff-terminates-normalize*:
$\Gamma\vdash_p \textit{normalize } c\downarrow s = \Gamma\vdash_p c\downarrow s$
by (*auto intro*: *terminates-to-terminates-normalize*
terminates-normalize-to-terminates)

7.3 Lemmas about *LanguageCon.strip-guards*

lemma *terminates-strip-guards-to-terminates*: $\bigwedge s.\ \Gamma\vdash_p \textit{strip-guards } F\ c\downarrow s \Longrightarrow \Gamma\vdash_p c\downarrow s$
proof (*induct c*)
case *Skip* **thus** *?case* **by** *simp*
next
case *Basic* **thus** *?case* **by** *simp*
next
case *Spec* **thus** *?case* **by** *simp*
next
case (*Seq c1 c2*)
hence $\Gamma\vdash_p \textit{Seq}\ (\textit{strip-guards } F\ c1)\ (\textit{strip-guards } F\ c2) \downarrow s$ **by** *simp*
thus $\Gamma\vdash_p \textit{Seq } c1\ c2 \downarrow s$
proof (*cases*)
fix *f* **assume** *s=Fault f* **thus** *?thesis* **by** *simp*
next
assume *s=Stuck* **thus** *?thesis* **by** *simp*
next
fix s' **assume** $s=\textit{Abrupt } s'$ **thus** *?thesis* **by** *simp*
next
fix s'
assume *s*: $s=\textit{Normal } s'$
assume $\Gamma\vdash_p \textit{strip-guards } F\ c1 \downarrow \textit{Normal } s'$
hence $\Gamma\vdash_p c1 \downarrow \textit{Normal } s'$
by (*rule Seq.hyps*)
moreover
assume *c2*:
$\forall s''.\ \Gamma\vdash_p\langle \textit{strip-guards } F\ c1, \textit{Normal } s'\rangle \Rightarrow s'' \longrightarrow \Gamma\vdash_p \textit{strip-guards } F\ c2\downarrow s''$
{
fix s'' **assume** *exec-c1*: $\Gamma\vdash_p\langle c1, \textit{Normal } s'\rangle \Rightarrow s''$
have $\Gamma\vdash_p c2 \downarrow s''$
proof (*cases* s'')
case ($\textit{Normal } s'''$)
with *exec-c1*

```
      have Γ⊢p⟨strip-guards F c1,Normal s' ⟩ ⇒ s''
        by (auto intro: exec-to-exec-strip-guards)
      with c2
      show ?thesis
        by (iprover intro: Seq.hyps)
    next
      case (Abrupt s''')
      with exec-c1
      have Γ⊢p⟨strip-guards F c1,Normal s' ⟩ ⇒ s''
        by (auto intro: exec-to-exec-strip-guards )
      with c2
      show ?thesis
        by (iprover intro: Seq.hyps)
    next
      case Fault thus ?thesis by simp
    next
      case Stuck thus ?thesis by simp
    qed
  }
  ultimately show ?thesis
    using s
    by (iprover intro: terminates.intros)
qed
next
  case (Cond b c1 c2)
  hence Γ⊢pCond b (strip-guards F c1) (strip-guards F c2) ↓ s by simp
  thus Γ⊢pCond b c1 c2 ↓ s
  proof (cases)
    fix f assume s=Fault f thus ?thesis by simp
  next
    assume s=Stuck thus ?thesis by simp
  next
    fix s' assume s=Abrupt s' thus ?thesis by simp
  next
    fix s'
    assume s'∈b Γ⊢pstrip-guards F c1 ↓ Normal s' s = Normal s'
    thus ?thesis
      by (iprover intro: terminates.intros Cond.hyps)
  next
    fix s'
    assume s'∉b Γ⊢pstrip-guards F c2 ↓ Normal s' s = Normal s'
    thus ?thesis
      by (iprover intro: terminates.intros Cond.hyps)
  qed
next
  case (While b c)
  have hyp-c: ⋀s. Γ⊢pstrip-guards F c ↓ s ⟹ Γ⊢pc ↓ s by fact
  have Γ⊢pWhile b (strip-guards F c) ↓ s using While.prems by simp
  moreover
```

```
  {
    fix sw
    assume Γ⊢p sw↓s
    then have sw=While b (strip-guards F c) ⟹
      Γ⊢p While b c ↓ s
    proof (induct)
      case (WhileTrue s b' c')
      have eqs: While b' c' = While b (strip-guards F c) by fact
      with ‹s∈b'› have b: s∈b by simp
      from eqs ‹Γ⊢p c' ↓ Normal s› have Γ⊢p strip-guards F c ↓ Normal s
        by simp
      hence term-c: Γ⊢p c ↓ Normal s
        by (rule hyp-c)
      moreover
      {
        fix t
        assume exec-c: Γ⊢p⟨c,Normal s ⟩ ⇒ t
        have Γ⊢p While b c ↓ t
        proof (cases t)
          case Fault
          thus ?thesis by simp
        next
          case Stuck
          thus ?thesis by simp
        next
          case (Abrupt t')
          thus ?thesis by simp
        next
          case (Normal t')
          with exec-c
          have Γ⊢p⟨strip-guards F c,Normal s ⟩ ⇒ Normal t'
            by (auto intro: exec-to-exec-strip-guards)
          with WhileTrue.hyps eqs Normal
          show ?thesis
            by fastforce
        qed
      }
      ultimately
      show ?case
        using b
        by (auto intro: terminates.intros)
    next
      case WhileFalse thus ?case by (auto intro: terminates.intros)
    qed simp-all
  }
  ultimately show Γ⊢p While b c ↓ s
    by auto
next
  case Call thus ?case by simp
```

next
  case *DynCom* **thus** *?case*
    by (*cases s*) (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros rangeI*)
next
  case *Guard*
  thus *?case*
    by (*cases s*) (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros*
              split: *if-split-asm*)
next
  case *Throw* **thus** *?case* **by** *simp*
next
  case (*Catch c1 c2*)
  hence $\Gamma\vdash_p$ *Catch* (*strip-guards F c1*) (*strip-guards F c2*) $\downarrow$ *s* **by** *simp*
  thus $\Gamma\vdash_p$ *Catch c1 c2* $\downarrow$ *s*
  proof (*cases*)
    fix *f* **assume** *s=Fault f* **thus** *?thesis* **by** *simp*
  next
    assume *s=Stuck* **thus** *?thesis* **by** *simp*
  next
    fix s' **assume** *s=Abrupt* s' **thus** *?thesis* **by** *simp*
  next
    fix s'
    assume *s*: *s=Normal* s'
    assume $\Gamma\vdash_p$ *strip-guards F c1* $\downarrow$ *Normal* s'
    hence $\Gamma\vdash_p$ *c1* $\downarrow$ *Normal* s'
      by (*rule Catch.hyps*)
    moreover
    assume *c2*:
      $\forall s''.$ $\Gamma\vdash_p\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow$ *Abrupt* s''
          $\longrightarrow$ $\Gamma\vdash_p$ *strip-guards F c2*$\downarrow$*Normal* s''
    {
      fix s'' **assume** *exec-c1*: $\Gamma\vdash_p\langle$*c1*,*Normal* $s'\rangle \Rightarrow$ *Abrupt* s''
      have $\Gamma\vdash_p$ *c2* $\downarrow$ *Normal* s''
      proof −
        from *exec-c1*
        have $\Gamma\vdash_p\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow$ *Abrupt* s''
          by (*auto intro*: *exec-to-exec-strip-guards*)
        with *c2*
        show *?thesis*
          by (*auto intro*: *Catch.hyps*)
      qed
    }
    ultimately show *?thesis*
      using *s*
      by (*iprover intro*: *terminates.intros*)
  qed
next case (*Await b c*) **thus** *?case*
  by (*cases s*) (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates-strip-guards-to-terminates*

terminates.intros
        split: *if-split-asm*)
qed

lemma *terminates-strip-to-terminates*:
  assumes *termi-strip*: *strip F* $\Gamma\vdash_p c\downarrow s$
  shows $\Gamma\vdash_p c\downarrow s$
using *termi-strip*
proof *induct*
  case (*Seq c1 s c2*)
  have $\Gamma\vdash_p c1 \downarrow Normal\ s$ **by** *fact*
  moreover
  {
    fix *s'*
    assume *exec*: $\Gamma\vdash_p \langle c1, Normal\ s\rangle \Rightarrow s'$
    have $\Gamma\vdash_p c2 \downarrow s'$
    proof (*cases isFault s'*)
      case *True*
      thus *?thesis*
        by (*auto elim*: *isFaultE*)
    next
      case *False*
      from *exec-to-exec-strip* [*OF exec this*] *Seq.hyps*
      show *?thesis*
        by *auto*
    qed
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
next
  case (*WhileTrue s b c*)
  have $\Gamma\vdash_p c \downarrow Normal\ s$ **by** *fact*
  moreover
  {
    fix *s'*
    assume *exec*: $\Gamma\vdash_p \langle c, Normal\ s\rangle \Rightarrow s'$
    have $\Gamma\vdash_p While\ b\ c \downarrow s'$
    proof (*cases isFault s'*)
      case *True*
      thus *?thesis*
        by (*auto elim*: *isFaultE*)
    next
      case *False*
      from *exec-to-exec-strip* [*OF exec this*] *WhileTrue.hyps*
      show *?thesis*
        by *auto*
    qed
  }
  ultimately show *?case*

    by (*auto intro*: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have $\Gamma\vdash_p c1 \downarrow Normal\ s$ **by** *fact*
  moreover
  {
    fix s'
    assume *exec*: $\Gamma\vdash_p \langle c1,Normal\ s\rangle \Rightarrow Abrupt\ s'$
    from *exec-to-exec-strip* [*OF exec*] *Catch.hyps*
    have $\Gamma\vdash_p c2 \downarrow Normal\ s'$
      by *auto*
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
next
  case *Call* **thus** *?case*
    by (*auto intro*: *terminates.intros terminates-strip-guards-to-terminates*)
next
  case (*AwaitTrue s b* Γ_p *c*)
  then have *eq-fun*:*Language.strip F* $(\Gamma_{\neg a}) = \Gamma_p$
    by (*simp add*: *AwaitTrue.hyps*(*2*) *strip-eq*)
  then have *Language.strip F* $(\Gamma_{\neg a})\vdash c \downarrow Normal\ s$ **using** *AwaitTrue.hyps*(*3*)
    by *auto*
  thus *?case* **by**
   (*fastforce intro*: *AwaitTrue.hyps*(*1*) *terminates.AwaitTrue terminates-strip-to-terminates*)

qed (*auto intro*: *terminates.intros*)

7.4 Lemmas about $c_1 \cap_g c_2$

lemma *inter-guards-terminates*:
  $\bigwedge c\ c2\ s.\ [\![(c1 \cap_{gs} c2) = Some\ c;\ \Gamma\vdash_p c1\downarrow s\ ]\!]$
      $\Longrightarrow \Gamma\vdash_p c\downarrow s$
proof (*induct c1*)
  case *Skip* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Skip*)
next
  case (*Basic f*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Basic*)
next
  case (*Spec r*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Spec*)
next
  case (*Seq a1 a2*)
  have $(Seq\ a1\ a2 \cap_{gs} c2) = Some\ c$ **by** *fact*
  then obtain *b1 b2 d1 d2* **where**
    c2: *c2=Seq b1 b2* **and**
    d1: $(a1 \cap_{gs} b1) = Some\ d1$ **and** *d2*: $(a2 \cap_{gs} b2) = Some\ d2$ **and**
    c: *c=Seq d1 d2*
    by (*auto simp add*: *inter-guards-Seq*)
  have *termi-c1*: $\Gamma\vdash_p Seq\ a1\ a2 \downarrow s$ **by** *fact*
  have $\Gamma\vdash_p Seq\ d1\ d2 \downarrow s$

```
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    note Normal-s = this
    with d1 termi-c1
    have Γ⊢p d1 ↓ Normal s'
      by (auto elim: terminates-Normal-elim-cases intro: Seq.hyps)
    moreover
    {
      fix t
      assume exec-d1: Γ⊢p⟨d1,Normal s' ⟩ ⇒ t
      have Γ⊢p d2 ↓ t
      proof (cases t)
        case Fault thus ?thesis by simp
      next
        case Stuck thus ?thesis by simp
      next
        case Abrupt thus ?thesis by simp
      next
        case (Normal t')
        with inter-guards-exec-noFault [OF d1 exec-d1]
        have Γ⊢p⟨a1,Normal s' ⟩ ⇒ Normal t'
          by simp
        with termi-c1 Normal-s have Γ⊢p a2 ↓ Normal t'
          by (auto elim: terminates-Normal-elim-cases)
        with d2 have Γ⊢p d2 ↓ Normal t'
          by (auto intro: Seq.hyps)
        with Normal show ?thesis by simp
      qed
    }
    ultimately have Γ⊢p Seq d1 d2 ↓ Normal s'
      by (fastforce intro: terminates.intros)
    with Normal show ?thesis by simp
  qed
  with c show ?case by simp
next
  case Cond thus ?case
    by − (cases s,
          auto intro: terminates.intros elim!: terminates-Normal-elim-cases
               simp add: inter-guards-Cond)
next
  case (While b bdy1)
  have (While b bdy1 ∩gs c2) = Some c by fact
  then obtain bdy2 bdy where
```

```
  c2: c2=While b bdy2 and
  bdy: (bdy1 ∩gs bdy2) = Some bdy and
  c: c=While b bdy
  by (auto simp add: inter-guards-While)
have Γ⊢p While b bdy1 ↓ s by fact
moreover
{
  fix s w w1 w2
  assume termi-w: Γ⊢p w ↓ s
  assume w: w=While b bdy1
  from termi-w w
  have Γ⊢p While b bdy ↓ s
  proof (induct)
    case (WhileTrue s b' bdy1')
    have eqs: While b' bdy1' = While b bdy1 by fact
    from WhileTrue have s-in-b: s ∈ b by simp
    from WhileTrue have termi-bdy1: Γ⊢p bdy1 ↓ Normal s by simp
    show ?case
    proof -
      from bdy termi-bdy1
      have Γ⊢p bdy↓(Normal s)
        by (rule While.hyps)
      moreover
      {
        fix t
        assume exec-bdy: Γ⊢p ⟨bdy,Normal s ⟩ ⇒ t
        have Γ⊢p While b bdy↓t
        proof (cases t)
          case Fault thus ?thesis by simp
        next
          case Stuck thus ?thesis by simp
        next
          case Abrupt thus ?thesis by simp
        next
          case (Normal t')
          with inter-guards-exec-noFault [OF bdy exec-bdy]
          have Γ⊢p ⟨bdy1,Normal s ⟩ ⇒ Normal t'
            by simp
          with WhileTrue have Γ⊢p While b bdy ↓ Normal t'
            by simp
          with Normal show ?thesis by simp
        qed
      }
      ultimately show ?thesis
        using s-in-b
        by (blast intro: terminates.WhileTrue)
    qed
  next
    case WhileFalse thus ?case
```

by (*blast intro*: *terminates.WhileFalse*)
qed (*simp-all*)
}
ultimately
show *?case* **using** *c* **by** *simp*
next
case *Call* **thus** *?case* **by** (*simp add*: *inter-guards-Call*)
next
case (*DynCom f1*)
have (*DynCom f1* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
then obtain *f2 f* **where**
c2: *c2=DynCom f2* **and**
f-defined: $\forall$ *s*. ((*f1 s*) $\cap_{gs}$ (*f2 s*)) $\neq$ *None* **and**
c: *c=DynCom* (λ*s*. *the* ((*f1 s*) $\cap_{gs}$ (*f2 s*)))
by (*auto simp add*: *inter-guards-DynCom*)
have *termi*: $\Gamma\vdash_p$*DynCom f1* $\downarrow$ *s* **by** *fact*
show *?case*
proof (*cases s*)
case *Fault* **thus** *?thesis* **by** *simp*
next
case *Stuck* **thus** *?thesis* **by** *simp*
next
case *Abrupt* **thus** *?thesis* **by** *simp*
next
case (*Normal s'*)
from *f-defined* **obtain** *f* **where** *f*: ((*f1 s'*) $\cap_{gs}$ (*f2 s'*)) = *Some f*
by *auto*
from *Normal termi*
have $\Gamma\vdash_p$*f1 s'*$\downarrow$ (*Normal s'*)
by (*auto elim*: *terminates-Normal-elim-cases*)
from *DynCom.hyps f this*
have $\Gamma\vdash_p$*f*$\downarrow$ (*Normal s'*)
by *blast*
with *c f Normal*
show *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*Guard f g1 bdy1*)
have (*Guard f g1 bdy1* $\cap_{gs}$ *c2*) = *Some c* **by** *fact*
then obtain *g2 bdy2 bdy* **where**
c2: *c2=Guard f g2 bdy2* **and**
bdy: (*bdy1* $\cap_{gs}$ *bdy2*) = *Some bdy* **and**
c: *c=Guard f* (*g1* $\cap$ *g2*) *bdy*
by (*auto simp add*: *inter-guards-Guard*)
have *termi-c1*: $\Gamma\vdash_p$*Guard f g1 bdy1* $\downarrow$ *s* **by** *fact*
show *?case*
proof (*cases s*)
case *Fault* **thus** *?thesis* **by** *simp*

```
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    show ?thesis
    proof (cases s' ∈ g1)
      case False
      with Normal c show ?thesis by (auto intro: terminates.GuardFault)
    next
      case True
      note s-in-g1 = this
      show ?thesis
      proof (cases s' ∈ g2)
        case False
        with Normal c show ?thesis by (auto intro: terminates.GuardFault)
      next
        case True
        with termi-c1 s-in-g1 Normal have Γ⊢p bdy1 ↓ Normal s'
          by (auto elim: terminates-Normal-elim-cases)
        with c bdy Guard.hyps Normal True s-in-g1
        show ?thesis by (auto intro: terminates.Guard)
      qed
    qed
  qed
next
  case Throw thus ?case
    by (auto simp add: inter-guards-Throw)
next
  case (Catch a1 a2)
  have (Catch a1 a2 ∩gs c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Catch b1 b2 and
    d1: (a1 ∩gs b1) = Some d1 and d2: (a2 ∩gs b2) = Some d2 and
    c: c=Catch d1 d2
    by (auto simp add: inter-guards-Catch)
  have termi-c1: Γ⊢p Catch a1 a2 ↓ s by fact
  have Γ⊢p Catch d1 d2 ↓ s
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    note Normal-s = this
    with d1 termi-c1
```

have $\Gamma\vdash_p d1 \downarrow Normal\ s'$
by (*auto elim*: *terminates-Normal-elim-cases intro*: *Catch.hyps*)
moreover
{
fix *t*
assume *exec-d1*: $\Gamma\vdash_p\langle d1,Normal\ s'\rangle \Rightarrow Abrupt\ t$
have $\Gamma\vdash_p d2 \downarrow Normal\ t$
proof −
from *inter-guards-exec-noFault* [*OF d1 exec-d1*]
have $\Gamma\vdash_p\langle a1,Normal\ s'\rangle \Rightarrow Abrupt\ t$
by *simp*
with *termi-c1 Normal-s* **have** $\Gamma\vdash_p a2 \downarrow Normal\ t$
by (*auto elim*: *terminates-Normal-elim-cases*)
with *d2* **have** $\Gamma\vdash_p d2 \downarrow Normal\ t$
by (*auto intro*: *Catch.hyps*)
with *Normal* **show** *?thesis* **by** *simp*
qed
}
ultimately have $\Gamma\vdash_p Catch\ d1\ d2 \downarrow Normal\ s'$
by (*fastforce intro*: *terminates.intros*)
with *Normal* **show** *?thesis* **by** *simp*
qed
with *c* **show** *?case* **by** *simp*
next
case (*Await b bdy1 e*)
have $(Await\ b\ bdy1\ e \cap_{gs} c2) = Some\ c$ **by** *fact*
then obtain *bdy2 bdy* **where**
c2: $c2{=}Await\ b\ bdy2\ e$ **and**
bdy: $(bdy1 \cap_g bdy2) = Some\ bdy$ **and**
c: $c{=}Await\ b\ bdy\ e$
by (*auto simp add*: *inter-guards-Await*)
have *termi-c1*:$\Gamma\vdash_p Await\ b\ bdy1\ e \downarrow s$ **by** *fact*
show *?case*
proof (*cases s*)
case *Fault* **thus** *?thesis* **by** *simp*
next
case *Stuck* **thus** *?thesis* **by** *simp*
next
case *Abrupt* **thus** *?thesis* **by** *simp*
next
case (*Normal s'*) **thus** *?thesis*
by (*metis* (*no-types*) *Await.prems*(*2*) *TerminationCon.terminates-Normal-elim-cases*(*12*) *bdy c*
inter-guards-terminates terminates.AwaitFalse terminates.AwaitTrue)
qed
qed

lemma *inter-guards-terminates'*:
assumes *c*: $(c1 \cap_{gs} c2) = Some\ c$

assumes *termi-c2*: $\Gamma\vdash_p c2\downarrow s$
shows $\Gamma\vdash_p c\downarrow s$
proof −
from *c* **have** $(c2 \cap_{gs} c1) = Some\ c$
by (*rule inter-guards-sym*)
from *this termi-c2* **show** *?thesis*
by (*rule inter-guards-terminates*)
qed

7.5 Lemmas about *LanguageCon.mark-guards*

lemma *terminates-to-terminates-mark-guards*:
assumes *termi*: $\Gamma\vdash_p c\downarrow s$
shows $\Gamma\vdash_p mark\text{-}guards\ f\ c\downarrow s$
using *termi*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Guard* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *Fault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case (*Seq c1 s c2*)
have $\Gamma\vdash_p mark\text{-}guards\ f\ c1 \downarrow Normal\ s$ **by** *fact*
moreover
{
fix *t*
assume *exec-mark*: $\Gamma\vdash_p\langle mark\text{-}guards\ f\ c1, Normal\ s\ \rangle \Rightarrow t$
have $\Gamma\vdash_p mark\text{-}guards\ f\ c2 \downarrow t$
proof −
from *exec-mark-guards-to-exec* [*OF exec-mark*] **obtain** t' **where**
exec-c1: $\Gamma\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow t'$ **and**
t-Fault: $isFault\ t \longrightarrow isFault\ t'$ **and**
t'*-Fault-f*: $t' = Fault\ f \longrightarrow t' = t$ **and**
t'*-Fault*: $isFault\ t' \longrightarrow isFault\ t$ **and**
t'*-noFault*: $\neg\ isFault\ t' \longrightarrow t' = t$
by *blast*
show *?thesis*
proof (*cases isFault* t')
case *True*
with t'*-Fault* **have** *isFault t* **by** *simp*
thus *?thesis*
by (*auto elim*: *isFaultE*)

```
        next
          case False
          with t'-noFault have t'=t by simp
          with exec-c1 Seq.hyps
          show ?thesis
            by auto
        qed
      qed
    }
    ultimately show ?case
      by (auto intro: terminates.intros)
  next
    case CondTrue thus ?case by (fastforce intro: terminates.intros)
  next
    case CondFalse thus ?case by (fastforce intro: terminates.intros)
  next
    case (WhileTrue s b c)
    have s-in-b: s ∈ b by fact
    have Γ⊢p mark-guards f c ↓ Normal s by fact
    moreover
    {
      fix t
      assume exec-mark: Γ⊢p ⟨mark-guards f c,Normal s ⟩ ⇒ t
      have Γ⊢p mark-guards f (While b c) ↓ t
      proof −
        from exec-mark-guards-to-exec [OF exec-mark] obtain t' where
          exec-c1: Γ⊢p ⟨c,Normal s ⟩ ⇒ t' and
          t-Fault: isFault t ⟶ isFault t' and
          t'-Fault-f: t' = Fault f ⟶ t' = t and
          t'-Fault: isFault t' ⟶ isFault t and
          t'-noFault: ¬ isFault t' ⟶ t' = t
          by blast
        show ?thesis
        proof (cases isFault t')
          case True
          with t'-Fault have isFault t by simp
          thus ?thesis
            by (auto elim: isFaultE)
        next
          case False
          with t'-noFault have t'=t by simp
          with exec-c1 WhileTrue.hyps
          show ?thesis
            by auto
        qed
      qed
    }
    ultimately show ?case
      by (auto intro: terminates.intros)
```

next
  case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Call* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *CallUndefined* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Stuck* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *DynCom* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Throw* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Abrupt* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have $\Gamma\vdash_p$*mark-guards f c1* $\downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix *t*
    assume *exec-mark*: $\Gamma\vdash_p\langle$*mark-guards f c1* ,*Normal s* $\rangle \Rightarrow$ *Abrupt t*
    have $\Gamma\vdash_p$*mark-guards f c2* $\downarrow$ *Normal t*
    proof −
      from *exec-mark-guards-to-exec* [*OF exec-mark*] **obtain** t' **where**
        exec-c1: $\Gamma\vdash_p\langle$*c1* ,*Normal s* $\rangle \Rightarrow t'$ **and**
        t'*-Fault-f*: t' = *Fault f* $\longrightarrow$ t' = *Abrupt t* **and**
        t'*-Fault*: *isFault* t' $\longrightarrow$ *isFault* (*Abrupt t*) **and**
        t'*-noFault*: $\neg$ *isFault* t' $\longrightarrow$ t' = *Abrupt t*
        by *fastforce*
      show *?thesis*
      proof (*cases isFault* t')
        case *True*
        with t'*-Fault* **have** *isFault* (*Abrupt t*) **by** *simp*
        thus *?thesis* **by** *simp*
      next
        case *False*
        with t'*-noFault* **have** t'=*Abrupt t* **by** *simp*
        with *exec-c1 Catch.hyps*
        show *?thesis*
          by *auto*
      qed
    qed
  }
  ultimately show *?case*
    by (*auto intro*: *terminates.intros*)
next
   case (*AwaitTrue s b* Γ_p *c*)
   then have $\Gamma_{\neg a}\vdash c \downarrow$ *Normal s*
  using *AwaitTrue.hyps*(*2*) *AwaitTrue.hyps*(*3*) *terminates-to-terminates-mark-guards*

by *blast*
  thus *?case*
  by (*simp add*: *AwaitTrue.hyps*(*1*) *terminates.AwaitTrue terminates-to-terminates-mark-guards*)

next
  case (*AwaitFalse s b*) **thus** *?case* **by** (*fastforce intro*: *terminates.AwaitFalse*)
qed

lemma *terminates-mark-guards-to-terminates-Normal*:
  $\bigwedge s.\ \Gamma\vdash_p$*mark-guards f c*$\downarrow$*Normal s* $\Longrightarrow$ $\Gamma\vdash_p c\downarrow$*Normal s*
proof (*induct c*)
  case *Skip* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Basic* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case (*Seq c1 c2*)
  have $\Gamma\vdash_p$*mark-guards f* (*Seq c1 c2*) $\downarrow$ *Normal s* **by** *fact*
  then obtain
    termi-merge-c1: $\Gamma\vdash_p$*mark-guards f c1* $\downarrow$ *Normal s* **and**
    termi-merge-c2: $\forall s'.\ \Gamma\vdash_p\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow s' \longrightarrow$
                $\Gamma\vdash_p$*mark-guards f c2* $\downarrow s'$
    by (*auto elim*: *terminates-Normal-elim-cases*)
  from *termi-merge-c1 Seq.hyps*
  have $\Gamma\vdash_p c1 \downarrow$ *Normal s* **by** *iprover*
  moreover
  {
    fix s'
    assume *exec-c1*: $\Gamma\vdash_p\langle c1$,*Normal s* $\rangle \Rightarrow s'$
    have $\Gamma\vdash_p\ c2 \downarrow s'$
    proof (*cases isFault s*$'$)
      case *True*
      thus *?thesis* **by** (*auto elim*: *isFaultE*)
    next
      case *False*
      from *exec-to-exec-mark-guards* [*OF exec-c1 False*]
      have $\Gamma\vdash_p\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow s'$ **.**
      from *termi-merge-c2* [*rule-format*, *OF this*] *Seq.hyps*
      show *?thesis*
        by (*cases s*$'$) (*auto*)
    qed
  }
  ultimately show *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Cond* **thus** *?case*
    by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next

case (*While b c*)
{
fix *u c'*
assume *termi-c'*: $\Gamma\vdash_p c' \downarrow$ *Normal u*
assume *c'*: $c' =$ *mark-guards f* (*While b c*)
have $\Gamma\vdash_p$ *While b c* $\downarrow$ *Normal u*
using *termi-c' c'*
proof (*induct*)
case (*WhileTrue s b' c'*)
have *s-in-b*: $s \in b$ **using** *WhileTrue* **by** *simp*
have $\Gamma\vdash_p$*mark-guards f c* $\downarrow$ *Normal s*
using *WhileTrue* **by** (*auto elim*: *terminates-Normal-elim-cases*)
with *While.hyps* **have** $\Gamma\vdash_p c \downarrow$ *Normal s*
by *auto*
moreover
have *hyp-w*: $\forall w.\ \Gamma\vdash_p\langle$*mark-guards f c*,*Normal s* $\rangle \Rightarrow w \longrightarrow \Gamma\vdash_p$ *While b c* $\downarrow$
w
using *WhileTrue* **by** *simp*
hence $\forall w.\ \Gamma\vdash_p\langle c,$*Normal s* $\rangle \Rightarrow w \longrightarrow \Gamma\vdash_p$ *While b c* $\downarrow w$
apply –
apply (*rule allI*)
apply (*case-tac w*)
apply (*auto dest*: *exec-to-exec-mark-guards*)
done
ultimately show *?case*
using *s-in-b*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
qed *auto*
}
with *While* **show** *?case* **by** *simp*
next
case *Call* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case (*Guard f g c*)
thus *?case* **by** (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *Throw* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case (*Catch c1 c2*)
have $\Gamma\vdash_p$*mark-guards f* (*Catch c1 c2*) $\downarrow$ *Normal s* **by** *fact*
then obtain
termi-merge-c1: $\Gamma\vdash_p$*mark-guards f c1* $\downarrow$ *Normal s* **and**

termi-merge-c2: $\forall s'.\ \Gamma\vdash_p\langle \textit{mark-guards } f\ c1,\textit{Normal } s\ \rangle \Rightarrow \textit{Abrupt } s' \longrightarrow$
$\Gamma\vdash_p \textit{mark-guards } f\ c2 \downarrow \textit{Normal } s'$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *termi-merge-c1 Catch.hyps*
have $\Gamma\vdash_p c1 \downarrow \textit{Normal } s$ **by** *iprover*
moreover
{
fix s'
assume *exec-c1*: $\Gamma\vdash_p\langle c1,\textit{Normal } s\ \rangle \Rightarrow \textit{Abrupt } s'$
have $\Gamma\vdash_p\ c2 \downarrow \textit{Normal } s'$
proof −
from *exec-to-exec-mark-guards* [*OF exec-c1*]
have $\Gamma\vdash_p\langle \textit{mark-guards } f\ c1,\textit{Normal } s\ \rangle \Rightarrow \textit{Abrupt } s'$ **by** *simp*
from *termi-merge-c2* [*rule-format*, *OF this*] *Catch.hyps*
show *?thesis*
by *iprover*
qed
}
ultimately show *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Await b c*) **thus** *?case*
using *terminates-mark-guards-to-terminates-Normal*
by (*fastforce intro*: *terminates.intros*(*11*) *terminates.intros*(*12*) *elim*: *terminates-Normal-elim-cases*)
qed

lemma *terminates-mark-guards-to-terminates*:
$\Gamma\vdash_p \textit{mark-guards } f\ c\downarrow s \Longrightarrow \Gamma\vdash_p c\downarrow\ s$
by (*cases s*) (*auto intro*: *terminates-mark-guards-to-terminates-Normal*)

7.6 Lemmas about *LanguageCon.merge-guards*

lemma *terminates-to-terminates-merge-guards*:
assumes *termi*: $\Gamma\vdash_p c\downarrow s$
shows $\Gamma\vdash_p \textit{merge-guards } c\downarrow s$
using *termi*
proof (*induct*)
case (*Guard s g c f*)
have *s-in-g*: $s \in g$ **by** *fact*
have *termi-merge-c*: $\Gamma\vdash_p \textit{merge-guards } c \downarrow \textit{Normal } s$ **by** *fact*
show *?case*
proof (*cases* $\exists f'\ g'\ c'.\ \textit{merge-guards } c = \textit{Guard } f'\ g'\ c'$)
case *False*
hence *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
by (*cases merge-guards c*) (*auto simp add*: *Let-def*)
with *s-in-g termi-merge-c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
next
case *True*
then obtain $f'\ g'\ c'$ **where**

mc: *merge-guards c* = *Guard f′ g′ c′*
by *blast*
show *?thesis*
proof (*cases f*=*f′*)
case *False*
with *mc* have *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
by (*simp add*: *Let-def*)
with *s-in-g termi-merge-c* show *?thesis*
by (*auto intro*: *terminates.intros*)
next
case *True*
with *mc* have *merge-guards* (*Guard f g c*) = *Guard f* ($g \cap g'$) *c′*
by *simp*
with *s-in-g mc True termi-merge-c*
show *?thesis*
by (*cases* $s \in g'$)
(*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
qed
qed
next
case (*GuardFault s g f c*)
have $s \notin g$ by *fact*
thus *?case*
by (*cases merge-guards c*)
(*auto intro*: *terminates.intros split*: *if-split-asm simp add*: *Let-def*)
next
case (*AwaitTrue s b* Γ*1 c*)
thus *?case*
by (*simp add*: *terminates-to-terminates-merge-guards terminates.AwaitTrue*)
qed (*fastforce intro*: *terminates.intros dest*: *exec-merge-guards-to-exec*)+

lemma *terminates-merge-guards-to-terminates-Normal*:
shows $\bigwedge s.\ \Gamma\vdash_p$*merge-guards c*↓*Normal s* ⟹ $\Gamma\vdash_p c$↓*Normal s*
proof (*induct c*)
case *Skip* thus *?case* by (*fastforce intro*: *terminates.intros*)
next
case *Basic* thus *?case* by (*fastforce intro*: *terminates.intros*)
next
case *Spec* thus *?case* by (*fastforce intro*: *terminates.intros*)
next
case (*Seq c1 c2*)
have $\Gamma\vdash_p$*merge-guards* (*Seq c1 c2*) ↓ *Normal s* by *fact*
then obtain
termi-merge-c1: $\Gamma\vdash_p$*merge-guards c1* ↓ *Normal s* and
termi-merge-c2: ∀ *s′*. $\Gamma\vdash_p$⟨*merge-guards c1*,*Normal s* ⟩ ⇒ *s′* ⟶
$\Gamma\vdash_p$*merge-guards c2* ↓ *s′*
by (*auto elim*: *terminates-Normal-elim-cases*)
from *termi-merge-c1 Seq.hyps*
have $\Gamma\vdash_p c1$ ↓ *Normal s* by *iprover*

```
  moreover
  {
    fix s'
    assume exec-c1: Γ⊢p⟨c1,Normal s ⟩ ⇒ s'
    have Γ⊢p c2 ↓ s'
    proof -
      from exec-to-exec-merge-guards [OF exec-c1]
      have Γ⊢p⟨merge-guards c1,Normal s ⟩ ⇒ s' .
      from termi-merge-c2 [rule-format, OF this] Seq.hyps
      show ?thesis
        by (cases s') (auto)
    qed
  }
  ultimately show ?case by (auto intro: terminates.intros)
next
  case Cond thus ?case
    by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (While b c)
  {
    fix u c'
    assume termi-c': Γ⊢p c' ↓ Normal u
    assume c': c' = merge-guards (While b c)
    have Γ⊢p While b c ↓ Normal u
      using termi-c' c'
    proof (induct)
      case (WhileTrue s b' c')
      have s-in-b: s ∈ b using WhileTrue by simp
      have Γ⊢p merge-guards c ↓ Normal s
        using WhileTrue by (auto elim: terminates-Normal-elim-cases)
      with While.hyps have Γ⊢p c ↓ Normal s
        by auto
      moreover
      have hyp-w: ∀ w. Γ⊢p⟨merge-guards c,Normal s ⟩ ⇒ w ⟶ Γ⊢p While b c ↓ w
        using WhileTrue by simp
      hence ∀ w. Γ⊢p⟨c,Normal s ⟩ ⇒ w ⟶ Γ⊢p While b c ↓ w
        by (simp add: exec-iff-exec-merge-guards [symmetric])
      ultimately show ?case
        using s-in-b
        by (auto intro: terminates.intros)
    next
      case WhileFalse thus ?case by (auto intro: terminates.intros)
    qed auto
  }
  with While show ?case by simp
next
  case Call thus ?case
    by (fastforce intro: terminates.intros )
next
```

case *DynCom* **thus** *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case (*Guard f g c*)
have *termi-merge*: $\Gamma\vdash_p$*merge-guards* (*Guard f g c*) $\downarrow$ *Normal s* **by** *fact*
show *?case*
proof (*cases* $\exists f'\ g'\ c'$. *merge-guards c* = *Guard* $f'\ g'\ c'$)
case *False*
hence *m*: *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
by (*cases merge-guards c*) (*auto simp add*: *Let-def*)
from *termi-merge Guard.hyps* **show** *?thesis*
by (*simp only*: *m*)
(*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *True*
then obtain $f'\ g'\ c'$ **where**
mc: *merge-guards c* = *Guard* $f'\ g'\ c'$
by *blast*
show *?thesis*
proof (*cases* $f{=}f'$)
case *False*
with *mc* **have** *m*: *merge-guards* (*Guard f g c*) = *Guard f g* (*merge-guards c*)
by (*simp add*: *Let-def*)
from *termi-merge Guard.hyps* **show** *?thesis*
by (*simp only*: *m*)
(*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *True*
with *mc* **have** *m*: *merge-guards* (*Guard f g c*) = *Guard f* $(g \cap g')$ c'
by *simp*
from *termi-merge Guard.hyps*
show *?thesis*
by (*simp only*: *m mc*)
(*auto intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
qed
qed
next
case *Throw* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case (*Catch c1 c2*)
have $\Gamma\vdash_p$*merge-guards* (*Catch c1 c2*) $\downarrow$ *Normal s* **by** *fact*
then obtain
termi-merge-c1: $\Gamma\vdash_p$*merge-guards c1* $\downarrow$ *Normal s* **and**
termi-merge-c2: $\forall s'$. $\Gamma\vdash_p\langle$*merge-guards c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* $s' \longrightarrow$
$\Gamma\vdash_p$*merge-guards c2* $\downarrow$ *Normal* s'
by (*auto elim*: *terminates-Normal-elim-cases*)
from *termi-merge-c1 Catch.hyps*
have $\Gamma\vdash_p$*c1* $\downarrow$ *Normal s* **by** *iprover*

moreover
{
fix s'
assume *exec-c1*: $\Gamma\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
have $\Gamma\vdash_p\ c2 \downarrow Normal\ s'$
proof −
from *exec-to-exec-merge-guards* [*OF exec-c1*]
have $\Gamma\vdash_p\langle merge\text{-}guards\ c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$ **.**
from *termi-merge-c2* [*rule-format*, *OF this*] *Catch.hyps*
show *?thesis*
by *iprover*
qed
}
ultimately show *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Await b c*) **thus** *?case*
using *terminates-merge-guards-to-terminates-Normal*
by (*fastforce intro*: *terminates.intros*(*11*) *terminates.intros*(*12*) *elim*: *terminates-Normal-elim-cases*)
qed

lemma *terminates-merge-guards-to-terminates*:
$\Gamma\vdash_p merge\text{-}guards\ c\downarrow\ s \Longrightarrow \Gamma\vdash_p c\downarrow\ s$
by (*cases s*) (*auto intro*: *terminates-merge-guards-to-terminates-Normal*)

theorem *terminates-iff-terminates-merge-guards*:
$\Gamma\vdash_p c\downarrow\ s = \Gamma\vdash_p merge\text{-}guards\ c\downarrow\ s$
by (*iprover intro*: *terminates-to-terminates-merge-guards*
terminates-merge-guards-to-terminates)

7.7 Lemmas about $c_1 \subseteq_g c_2$

lemma *terminates-fewer-guards-Normal*:
shows $\bigwedge c\ s.\ [\![\Gamma\vdash_p c'\downarrow Normal\ s;\ c \subseteq_{gs} c';\ \Gamma\vdash_p\langle c', Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV]\!]$
$\Longrightarrow \Gamma\vdash_p c\downarrow Normal\ s$
proof (*induct c'*)
case *Skip* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
case *Basic* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
case *Spec* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
case (*Seq c1' c2'*)
have *termi*: $\Gamma\vdash_p Seq\ c1'\ c2' \downarrow Normal\ s$ **by** *fact*
then obtain
termi-c1': $\Gamma\vdash_p c1'\downarrow Normal\ s$ **and**
termi-c2': $\forall s'.\ \Gamma\vdash_p\langle c1', Normal\ s\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p c2'\downarrow s'$
by (*auto elim*: *terminates-Normal-elim-cases*)
have *noFault*: $\Gamma\vdash_p\langle Seq\ c1'\ c2', Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
hence *noFault-c1'*: $\Gamma\vdash_p\langle c1', Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$

by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
have $c \subseteq_{gs}$ *Seq c1' c2'* **by** *fact*
from *subseteq-guards-Seq* [*OF this*] **obtain** *c1 c2* **where**
  c: *c* = *Seq c1 c2* **and**
  c1-c1': $c1 \subseteq_{gs} c1'$ **and**
  c2-c2': $c2 \subseteq_{gs} c2'$
  by *blast*
from *termi-c1' c1-c1' noFault-c1'*
have $\Gamma\vdash_p c1\downarrow$ *Normal s*
  by (*rule Seq.hyps*)
moreover
{
  fix *t*
  assume *exec-c1*: $\Gamma\vdash_p\langle c1,Normal\ s\ \rangle \Rightarrow t$
  have $\Gamma\vdash_p c2\downarrow t$
  proof −
    from *exec-to-exec-subseteq-guards* [*OF c1-c1' exec-c1*] **obtain** *t'* **where**
      exec-c1': $\Gamma\vdash_p\langle c1',Normal\ s\ \rangle \Rightarrow t'$ **and**
      t-Fault: *isFault t* $\longrightarrow$ *isFault t'* **and**
      t'-noFault: $\neg$ *isFault t'* $\longrightarrow$ *t'* = *t*
      by *blast*
    show *?thesis*
    proof (*cases isFault t'*)
      case *True*
      with *exec-c1' noFault-c1'*
      have *False*
        by (*fastforce elim*: *isFaultE dest*: *Fault-end simp add*: *final-notin-def*)
      thus *?thesis* **..**
    next
      case *False*
      with *t'-noFault* **have** *t'*: *t'*=*t* **by** *simp*
      with *termi-c2' exec-c1'*
      have *termi-c2'*: $\Gamma\vdash_p c2'\downarrow t$
        by *auto*
      show *?thesis*
      proof (*cases t*)
        case *Fault* **thus** *?thesis* **by** *auto*
      next
        case *Abrupt* **thus** *?thesis* **by** *auto*
      next
        case *Stuck* **thus** *?thesis* **by** *auto*
      next
        case (*Normal u*)
        with *noFault exec-c1' t'*
        have $\Gamma\vdash_p\langle c2',Normal\ u\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*
          by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
        from *termi-c2'* [*simplified Normal*] *c2-c2' this*
        have $\Gamma\vdash_p c2 \downarrow$ *Normal u*
          by (*rule Seq.hyps*)

with *Normal exec-c1*
show *?thesis* **by** *simp*
qed
qed
qed
}
ultimately show *?case* **using** *c* **by** (*auto intro*: *terminates.intros*)
next
case (*Cond b c1′ c2′*)
have *noFault*: $\Gamma\vdash_p\langle Cond\ b\ c1'\ c2',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
have *termi*: $\Gamma\vdash_p Cond\ b\ c1'\ c2' \downarrow Normal\ s$ **by** *fact*
have $c \subseteq_{gs} Cond\ b\ c1'\ c2'$ **by** *fact*
from *subseteq-guards-Cond* [*OF this*] **obtain** *c1 c2* **where**
c: $c = Cond\ b\ c1\ c2$ **and**
c1-c1′: $c1 \subseteq_{gs} c1'$ **and**
c2-c2′: $c2 \subseteq_{gs} c2'$
by *blast*
thus *?case*
proof (*cases* $s \in b$)
case *True*
with *termi* **have** *termi-c1′*: $\Gamma\vdash_p c1' \downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *True noFault* **have** $\Gamma\vdash_p\langle c1',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c1′ c1-c1′ this*
have $\Gamma\vdash_p c1 \downarrow Normal\ s$
by (*rule Cond.hyps*)
with *True c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
next
case *False*
with *termi* **have** *termi-c2′*: $\Gamma\vdash_p c2' \downarrow Normal\ s$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *False noFault* **have** $\Gamma\vdash_p\langle c2',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c2′ c2-c2′ this*
have $\Gamma\vdash_p c2 \downarrow Normal\ s$
by (*rule Cond.hyps*)
with *False c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*While b c′*)
have *noFault*: $\Gamma\vdash_p\langle While\ b\ c',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
have *termi*: $\Gamma\vdash_p While\ b\ c' \downarrow Normal\ s$ **by** *fact*
have $c \subseteq_{gs} While\ b\ c'$ **by** *fact*
from *subseteq-guards-While* [*OF this*]
obtain *c″* **where**
c: $c = While\ b\ c''$ **and**

c''-c': $c'' \subseteq_{gs}$ c'
by *blast*
{
fix *d u*
assume *termi*: $\Gamma\vdash_p d \downarrow u$
assume *d*: $d = While\ b\ c'$
assume *noFault*: $\Gamma\vdash_p\langle While\ b\ c',u\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*
have $\Gamma\vdash_p While\ b\ c'' \downarrow u$
using *termi d noFault*
proof (*induct*)
case (*WhileTrue u b' c'''*)
have *u-in-b*: $u \in b$ **using** *WhileTrue* **by** *simp*
have *termi-c'*: $\Gamma\vdash_p c' \downarrow Normal\ u$ **using** *WhileTrue* **by** *simp*
have *noFault*: $\Gamma\vdash_p\langle While\ b\ c',Normal\ u\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV* **using** *WhileTrue*
by *simp*
hence *noFault-c'*: $\Gamma\vdash_p\langle c',Normal\ u\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV* **using** *u-in-b*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *While.hyps* [*OF termi-c' c''-c' this*]
have $\Gamma\vdash_p c'' \downarrow Normal\ u$**.**
moreover
from *WhileTrue*
have *hyp-w*: $\forall s'.\ \Gamma\vdash_p\langle c',Normal\ u\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash_p\langle While\ b\ c',s'\ \rangle \Rightarrow\notin Fault$
‘ *UNIV*
$\longrightarrow \Gamma\vdash_p While\ b\ c'' \downarrow s'$
by *simp*
{
fix *v*
assume *exec-c''*: $\Gamma\vdash_p\langle c'',Normal\ u\ \rangle \Rightarrow v$
have $\Gamma\vdash_p While\ b\ c'' \downarrow v$
proof −
from *exec-to-exec-subseteq-guards* [*OF c''-c' exec-c''*] **obtain** v' **where**
exec-c': $\Gamma\vdash_p\langle c',Normal\ u\ \rangle \Rightarrow v'$ **and**
v-Fault: *isFault* $v \longrightarrow$ *isFault* v' **and**
v'*-noFault*: $\neg$ *isFault* $v' \longrightarrow v' = v$
by *auto*
show *?thesis*
proof (*cases isFault* v')
case *True*
with *exec-c' noFault u-in-b*
have *False*
by (*fastforce*
simp add: *final-notin-def intro*: *exec.intros elim*: *isFaultE*)
thus *?thesis* **..**
next
case *False*
with v'*-noFault* **have** v': $v'=v$
by *simp*
with *noFault exec-c' u-in-b*
have $\Gamma\vdash_p\langle While\ b\ c',v\ \rangle \Rightarrow\notin Fault$ ‘ *UNIV*

by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
from *hyp-w* [*rule-format*, *OF exec-c'* [*simplified v'*] *this*]
show $\Gamma\vdash_p$ *While b c''* $\downarrow$ *v* **.**
qed
qed
}
ultimately
show *?case* **using** *u-in-b*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
qed *auto*
}
with *c noFault termi* **show** *?case*
by *auto*
next
case *Call* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
case (*DynCom C'*)
have *termi*: $\Gamma\vdash_p$*DynCom C'* $\downarrow$ *Normal s* **by** *fact*
hence *termi-C'*: $\Gamma\vdash_p$*C' s* $\downarrow$ *Normal s*
by *cases*
have *noFault*: $\Gamma\vdash_p\langle$*DynCom C'*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *UNIV* **by** *fact*
hence *noFault-C'*: $\Gamma\vdash_p\langle$*C' s*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *UNIV*
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
have *c* $\subseteq_{gs}$ *DynCom C'* **by** *fact*
from *subseteq-guards-DynCom* [*OF this*] **obtain** *C* **where**
c: *c* = *DynCom C* **and**
C-C': $\forall$ *s*. *C s* $\subseteq_{gs}$ *C' s*
by *blast*
from *DynCom.hyps termi-C' C-C'* [*rule-format*] *noFault-C'*
have $\Gamma\vdash_p$*C s* $\downarrow$ *Normal s*
by *fast*
with *c* **show** *?case*
by (*auto intro*: *terminates.intros*)
next
case (*Guard f' g' c'*)
have *noFault*: $\Gamma\vdash_p\langle$*Guard f' g' c'*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *UNIV* **by** *fact*
have *termi*: $\Gamma\vdash_p$*Guard f' g' c'* $\downarrow$ *Normal s* **by** *fact*
have *c* $\subseteq_{gs}$ *Guard f' g' c'* **by** *fact*
hence *c-cases*: (*c* $\subseteq_{gs}$ *c'*) $\vee$ ($\exists$ *c''*. *c* = *Guard f' g' c''* $\wedge$ (*c''* $\subseteq_{gs}$ *c'*))
by (*rule subseteq-guards-Guard*)
thus *?case*
proof (*cases s* $\in$ *g'*)
case *True*
note *s-in-g'* = *this*
with *noFault* **have** *noFault-c'*: $\Gamma\vdash_p\langle$*c'*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *UNIV*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
from *termi s-in-g'* **have** *termi-c'*: $\Gamma\vdash_p$*c'* $\downarrow$ *Normal s*

    by *cases auto*
  from *c-cases* **show** *?thesis*
  proof
    assume $c \subseteq_{gs} c'$
    from *termi-c' this noFault-c'*
    show $\Gamma\vdash_p c \downarrow$ *Normal s*
      by (*rule Guard.hyps*)
  next
    assume $\exists\, c''.\ c = Guard\ f'\ g'\ c'' \wedge (c'' \subseteq_{gs} c')$
    then obtain c'' **where**
      c: $c = Guard\ f'\ g'\ c''$ **and** *c''-c'*: $c'' \subseteq_{gs} c'$
      by *blast*
    from *termi-c' c''-c' noFault-c'*
    have $\Gamma\vdash_p c'' \downarrow$ *Normal s*
      by (*rule Guard.hyps*)
    with *s-in-g' c*
    show *?thesis*
      by (*auto intro*: *terminates.intros*)
  qed
next
  case *False*
  with *noFault* **have** *False*
    by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
  thus *?thesis* **..**
qed
next
 case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros dest*: *subseteq-guardsD*)
next
 case (*Catch c1' c2'*)
 have *termi*: $\Gamma\vdash_p$ *Catch c1' c2'* $\downarrow$ *Normal s* **by** *fact*
 then obtain
   termi-c1': $\Gamma\vdash_p c1' \downarrow$ *Normal s* **and**
   termi-c2': $\forall s'.\ \Gamma\vdash_p\langle c1',Normal\ s\ \rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash_p c2' \downarrow Normal\ s'$
   by (*auto elim*: *terminates-Normal-elim-cases*)
 have *noFault*: $\Gamma\vdash_p\langle Catch\ c1'\ c2',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
 hence *noFault-c1'*: $\Gamma\vdash_p\langle c1',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
   by (*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
 have $c \subseteq_{gs} Catch\ c1'\ c2'$ **by** *fact*
 from *subseteq-guards-Catch* [*OF this*] **obtain** *c1 c2* **where**
   c: $c = Catch\ c1\ c2$ **and**
   c1-c1': $c1 \subseteq_{gs} c1'$ **and**
   c2-c2': $c2 \subseteq_{gs} c2'$
   by *blast*
 from *termi-c1' c1-c1' noFault-c1'*
 have $\Gamma\vdash_p c1 \downarrow$ *Normal s*
   by (*rule Catch.hyps*)
 moreover
 {
   fix *t*

assume *exec-c1*: $\Gamma\vdash_p\langle c1,Normal\ s\ \rangle \Rightarrow Abrupt\ t$
have $\Gamma\vdash_p c2\downarrow\ Normal\ t$
proof −
from *exec-to-exec-subseteq-guards* [*OF c1-c1′ exec-c1*] **obtain** t' **where**
exec-c1′: $\Gamma\vdash_p\langle c1',Normal\ s\ \rangle \Rightarrow t'$ **and**
t′-noFault: $\neg\ isFault\ t' \longrightarrow t' = Abrupt\ t$
by *blast*
show *?thesis*
proof (*cases isFault t′*)
case *True*
with *exec-c1′ noFault-c1′*
have *False*
by (*fastforce elim*: *isFaultE dest*: *Fault-end simp add*: *final-notin-def*)
thus *?thesis* **..**
next
case *False*
with *t′-noFault* **have** *t′*: $t'=Abrupt\ t$ **by** *simp*
with *termi-c2′ exec-c1′*
have *termi-c2′*: $\Gamma\vdash_p c2'\downarrow\ Normal\ t$
by *auto*
with *noFault exec-c1′ t′*
have $\Gamma\vdash_p\langle c2',Normal\ t\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c2′ c2-c2′ this*
show $\Gamma\vdash_p c2\ \downarrow\ Normal\ t$
by (*rule Catch.hyps*)
qed
qed
}
ultimately show *?case* **using** *c* **by** (*auto intro*: *terminates.intros*)
next
case (*Await b c′ e*)
have *noFault*: $\Gamma\vdash_p\langle Await\ b\ c'\ e,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
have *termi*: $\Gamma\vdash_p Await\ b\ c'\ e\ \downarrow\ Normal\ s$ **by** *fact*
have $c \subseteq_{gs}\ Await\ b\ c'\ e$ **by** *fact*
from *subseteq-guards-Await* [*OF this*]
obtain c'' **where**
c: $c = Await\ b\ c''\ e$ **and**
c′′-c′: $c'' \subseteq_g\ c'$
by *blast*
with *c c′′-c′ noFault termi*
show *?case* **using** *terminates-fewer-guards-Normal*
by (*metis Semantic.final-notinI SemanticCon.final-notin-def TerminationCon.terminates-Normal-elim-cases exec.AwaitTrue terminates.AwaitFalse terminates.AwaitTrue*)

qed

theorem *terminates-fewer-guards*:
shows $[\![\Gamma\vdash_p c'\downarrow s;\ c \subseteq_{gs}\ c';\ \Gamma\vdash_p\langle c',s\ \rangle \Rightarrow\notin Fault\ `\ UNIV]\!]$

```
      ⟹ Γ⊢p c↓s
  by (cases s) (auto intro: terminates-fewer-guards-Normal)

lemma terminates-noFault-strip-guards:
  assumes termi: Γ⊢p c↓Normal s
  shows ⟦Γ⊢p⟨c,Normal s ⟩ ⇒∉Fault ‘ F⟧ ⟹ Γ⊢p strip-guards F c↓Normal s
using termi
proof (induct)
  case Skip thus ?case by (auto intro: terminates.intros)
next
  case Basic thus ?case by (auto intro: terminates.intros)
next
  case Spec thus ?case by (auto intro: terminates.intros)
next
  case (Guard s g c f)
  have s-in-g: s ∈ g by fact
  have Γ⊢p c ↓ Normal s by fact
  have Γ⊢p⟨Guard f g c,Normal s ⟩ ⇒∉Fault ‘ F by fact
  with s-in-g have Γ⊢p⟨c,Normal s ⟩ ⇒∉Fault ‘ F
    by (fastforce simp add: final-notin-def intro: exec.intros)
  with Guard.hyps have Γ⊢p strip-guards F c ↓ Normal s by simp
  with s-in-g show ?case
    by (auto intro: terminates.intros)
next
  case GuardFault thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case Fault thus ?case by (auto intro: terminates.intros)
next
  case (Seq c1 s c2)
  have noFault-Seq: Γ⊢p⟨Seq c1 c2,Normal s ⟩ ⇒∉Fault ‘ F by fact
  hence noFault-c1: Γ⊢p⟨c1,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto simp add: final-notin-def intro: exec.intros)
  with Seq.hyps have Γ⊢p strip-guards F c1 ↓ Normal s by simp
  moreover
  {
    fix s'
    assume exec-strip-guards-c1: Γ⊢p⟨strip-guards F c1,Normal s ⟩ ⇒ s'
    have Γ⊢p strip-guards F c2 ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-guards-to-exec [OF exec-strip-guards-c1] noFault-c1
      have Γ⊢p⟨c1,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      moreover
      from this noFault-Seq have Γ⊢p⟨c2,s' ⟩ ⇒∉Fault ‘ F
```

```
        by (auto simp add: final-notin-def intro: exec.intros)
      ultimately show ?thesis
        using Seq.hyps by simp
    qed
  }
  ultimately show ?case
    by (auto intro: terminates.intros)
next
  case CondTrue thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case CondFalse thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case (WhileTrue s b c)
  have s-in-b: s ∈ b by fact
  have noFault-while: Γ⊢p ⟨While b c,Normal s ⟩ ⇒∉Fault ‘ F by fact
  with s-in-b have noFault-c: Γ⊢p ⟨c,Normal s ⟩ ⇒∉Fault ‘ F
    by (auto simp add: final-notin-def intro: exec.intros)
  with WhileTrue.hyps have Γ⊢p strip-guards F c ↓ Normal s by simp
  moreover
  {
    fix s'
    assume exec-strip-guards-c: Γ⊢p ⟨strip-guards F c,Normal s ⟩ ⇒ s'
    have Γ⊢p strip-guards F (While b c) ↓ s'
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-guards-to-exec [OF exec-strip-guards-c] noFault-c
      have Γ⊢p ⟨c,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      moreover
      from this s-in-b noFault-while have Γ⊢p ⟨While b c,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      ultimately show ?thesis
        using WhileTrue.hyps by simp
    qed
  }
  ultimately show ?case
    using WhileTrue.hyps by (auto intro: terminates.intros)
next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
next
  case Call thus ?case by (auto intro: terminates.intros)
next
  case CallUndefined thus ?case by (auto intro: terminates.intros)
next
```

case *Stuck* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case*
by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Abrupt* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Catch c1 s c2*)
have *noFault-Catch*: $\Gamma \vdash_p \langle Catch\ c1\ c2, Normal\ s\ \rangle \Rightarrow \notin Fault\ `\ F$ **by** *fact*
hence *noFault-c1*: $\Gamma \vdash_p \langle c1, Normal\ s\ \rangle \Rightarrow \notin Fault\ `\ F$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
with *Catch.hyps* **have** $\Gamma \vdash_p strip\text{-}guards\ F\ c1 \downarrow Normal\ s$ **by** *simp*
moreover
{
fix s'
assume *exec-strip-guards-c1*: $\Gamma \vdash_p \langle strip\text{-}guards\ F\ c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
have $\Gamma \vdash_p strip\text{-}guards\ F\ c2 \downarrow Normal\ s'$
proof −
from *exec-strip-guards-to-exec* [*OF exec-strip-guards-c1*] *noFault-c1*
have $\Gamma \vdash_p \langle c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this noFault-Catch* **have** $\Gamma \vdash_p \langle c2, Normal\ s'\ \rangle \Rightarrow \notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *Catch.hyps* **by** *simp*
qed
}
ultimately show *?case*
using *Catch.hyps* **by** (*auto intro*: *terminates.intros*)
next
case (*AwaitTrue s b* Γ_p *c*)
with *terminates-noFault-strip-guards*
have $\Gamma_p \vdash Language.strip\text{-}guards\ F\ c \downarrow Normal\ s$
by (*simp add*: *terminates-noFault-strip-guards Semantic.final-notinI SemanticCon.final-notin-def exec.AwaitTrue*)
thus *?case*
by (*simp add*: *AwaitTrue.hyps*(*1*) *AwaitTrue.hyps*(*2*) *terminates.AwaitTrue*)

next
case (*AwaitFalse s b*) **thus** *?case* **by** (*simp add*: *terminates.AwaitFalse*)
qed

7.8 Lemmas about *LanguageCon.strip-guards*

lemma *terminates-noFault-strip*:
assumes *termi*: $\Gamma \vdash_p c \downarrow Normal\ s$

shows $[\![\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F]\!] \Longrightarrow strip\ F\ \Gamma\vdash_p c\downarrow Normal\ s$
using *termi*
proof (*induct*)
  case *Skip* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Basic* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case (*Guard s g c f*)
  have *s-in-g*: $s \in g$ **by** *fact*
  have $\Gamma\vdash_p\langle Guard\ f\ g\ c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
  with *s-in-g* **have** $\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
    by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
  then have $strip\ F\ \Gamma\vdash_p c \downarrow Normal\ s$ **by** (*simp add*: *Guard.hyps*)
  with *s-in-g* **show** *?case*
    by (*auto intro*: *terminates.intros simp del*: *strip-simp*)
next
  case *GuardFault* **thus** *?case*
    by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
  case *Fault* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case (*Seq c1 s c2*)
  have *noFault-Seq*: $\Gamma\vdash_p\langle Seq\ c1\ c2,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
  hence *noFault-c1*: $\Gamma\vdash_p\langle c1,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
    by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
  then have $strip\ F\ \Gamma\vdash_p c1 \downarrow Normal\ s$ **by** (*simp add*: *Seq.hyps*)
  moreover
  {
    fix s'
    assume *exec-strip-c1*: $strip\ F\ \Gamma\vdash_p\langle c1,Normal\ s\ \rangle \Rightarrow s'$
    have $strip\ F\ \Gamma\vdash_p c2 \downarrow s'$
    proof (*cases isFault* s')
      case *True*
      thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
    next
      case *False*
      with *exec-strip-to-exec* [*OF exec-strip-c1*] *noFault-c1*
      have $\Gamma\vdash_p\langle c1,Normal\ s\ \rangle \Rightarrow s'$
        by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
      moreover
      from *this noFault-Seq* **have** $\Gamma\vdash_p\langle c2,s'\ \rangle \Rightarrow\notin Fault\ `\ F$
        by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
      ultimately show *?thesis*
        using *Seq.hyps* **by** (*simp del*: *strip-simp*)
    qed
  }
  ultimately show *?case*

by (*fastforce intro*: *terminates.intros*)
next
case *CondTrue* **thus** *?case*
by (*fastforce intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *CondFalse* **thus** *?case*
by (*fastforce intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case (*WhileTrue s b c*)
have *s-in-b*: $s \in b$ **by** *fact*
have *noFault-while*: $\Gamma\vdash_p\langle While\ b\ c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
with *s-in-b* **have** *noFault-c*: $\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
then have *strip F* $\Gamma\vdash_p c \downarrow Normal\ s$ **by** (*simp add*: *WhileTrue.hyps*)
moreover
{
fix s'
assume *exec-strip-c*: *strip F* $\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow\ s'$
have *strip F* $\Gamma\vdash_p While\ b\ c \downarrow s'$
proof (*cases isFault* s')
case *True*
thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
next
case *False*
with *exec-strip-to-exec* [*OF exec-strip-c*] *noFault-c*
have $\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow\ s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
moreover
from *this s-in-b noFault-while* **have** $\Gamma\vdash_p\langle While\ b\ c,s'\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
ultimately show *?thesis*
using *WhileTrue.hyps* **by** (*simp del*: *strip-simp*)
qed
}
ultimately show *?case*
using *WhileTrue.hyps* **by** (*auto intro*: *terminates.intros simp del*: *strip-simp*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Call p bdy s*)
have *bdy*: $\Gamma\ p = Some\ bdy$ **by** *fact*
have $\Gamma\vdash_p\langle Call\ p,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
with *bdy* **have** *bdy-noFault*: $\Gamma\vdash_p\langle bdy,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
then have *strip-bdy-noFault*: *strip F* $\Gamma\vdash_p\langle bdy,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def dest*!: *exec-strip-to-exec elim*!: *isFaultE*)

from *bdy-noFault* **have** *strip F* $\Gamma\vdash_p bdy \downarrow Normal\ s$ **by** (*simp add*: *Call.hyps*)
from *terminates-noFault-strip-guards* [*OF this strip-bdy-noFault*]

have *strip F* $\Gamma\vdash_p$*strip-guards F bdy* $\downarrow$ *Normal s*.
with *bdy* **show** *?case*
  by (*fastforce intro*: *terminates.Call*)
next
  case *CallUndefined* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Stuck* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *DynCom* **thus** *?case*
    by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
  case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case *Abrupt* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have *noFault-Catch*: $\Gamma\vdash_p\langle$*Catch c1 c2*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F* **by** *fact*
  hence *noFault-c1*: $\Gamma\vdash_p\langle$*c1*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ‘ *F*
    by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
  then have *strip F* $\Gamma\vdash_p$*c1* $\downarrow$ *Normal s* **by** (*simp add*: *Catch.hyps*)
  moreover
  {
    fix s'
    assume *exec-strip-c1*: *strip F* $\Gamma\vdash_p\langle$*c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* s'
    have *strip F* $\Gamma\vdash_p$*c2* $\downarrow$ *Normal* s'
    proof −
      from *exec-strip-to-exec* [*OF exec-strip-c1*] *noFault-c1*
      have $\Gamma\vdash_p\langle$*c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* s'
        by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
      moreover
      from *this noFault-Catch* **have** $\Gamma\vdash_p\langle$*c2*,*Normal* s' $\rangle \Rightarrow\notin$*Fault* ‘ *F*
        by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
      ultimately show *?thesis*
        using *Catch.hyps* **by** (*simp del*: *strip-simp*)
    qed
  }
  ultimately show *?case*
    using *Catch.hyps* **by** (*auto intro*: *terminates.intros simp del*: *strip-simp*)
next
  case (*AwaitTrue s b* Γ_p *c*)
  with *terminates-noFault-strip* **have** *Language.strip F* $\Gamma_p\vdash$ *c* $\downarrow$ *Normal s*
  by (*simp add*: *terminates-noFault-strip Semantic.final-notinI SemanticCon.final-notin-def exec.AwaitTrue*)
  then have *Language.strip F* Γ_p = (*LanguageCon.strip F* $\Gamma)_{\neg a}$
    by (*simp add*: *AwaitTrue.hyps*(*2*) *strip-eq*)
  then have (*LanguageCon.strip F* $\Gamma)_{\neg a}\vdash$*c* $\downarrow$ *Normal s*
    using ‹*Language.strip F* Γ_p = (*LanguageCon.strip F* $\Gamma)_{\neg a}$› ‹*Language.strip F* $\Gamma_p\vdash$*c* $\downarrow$ *Normal s*›
  by *presburger*

thus *?case*
by (*meson AwaitTrue.hyps*(*1*) *terminates.AwaitTrue*)
next
case(*AwaitFalse s b*) **thus** *?case* **by** (*simp add*:*terminates.AwaitFalse*)
qed

7.9 Miscellaneous

lemma *terminates-while-lemma*:
assumes *termi*: $\Gamma\vdash_p w\downarrow fk$
shows $\bigwedge k\ b\ c.\ [\![fk = Normal\ (f\ k);\ w{=}While\ b\ c;$
$\forall\, i.\ \Gamma\vdash_p\langle c,Normal\ (f\ i)\ \rangle \Rightarrow Normal\ (f\ (Suc\ i))]\!]$
$\Longrightarrow \exists\, i.\ f\ i \notin b$
using *termi*
proof (*induct*)
case *WhileTrue* **thus** *?case* **by** *blast*
next
case *WhileFalse* **thus** *?case* **by** *blast*
qed *simp-all*

lemma *terminates-while*:
$[\![\Gamma\vdash_p(While\ b\ c)\downarrow Normal\ (f\ k);$
$\forall\, i.\ \Gamma\vdash_p\langle c,Normal\ (f\ i)\ \rangle \Rightarrow Normal\ (f\ (Suc\ i))]\!]$
$\Longrightarrow \exists\, i.\ f\ i \notin b$
by (*blast intro*: *terminates-while-lemma*)

lemma *wf-terminates-while*:
$wf\ \{(t,s).\ \Gamma\vdash_p(While\ b\ c)\downarrow Normal\ s \wedge s{\in}b \wedge$
$\Gamma\vdash_p\langle c,Normal\ s\ \rangle \Rightarrow Normal\ t\}$
apply(*subst wf-iff-no-infinite-down-chain*)
apply(*rule notI*)
apply *clarsimp*
apply(*insert terminates-while*)
apply *blast*
done

lemma *terminates-restrict-to-terminates*:
assumes *terminates-res*: $\Gamma|_M\vdash_p\ c \downarrow s$
assumes *not-Stuck*: $\Gamma|_M\vdash_p\langle c,s\ \rangle \Rightarrow\notin\{Stuck\}$
shows $\Gamma\vdash_p\ c \downarrow s$
using *terminates-res not-Stuck*
proof (*induct*)
case *Skip* **show** *?case* **by** (*rule terminates.Skip*)
next
case *Basic* **show** *?case* **by** (*rule terminates.Basic*)
next
case *Spec* **show** *?case* **by** (*rule terminates.Spec*)
next
case *Guard* **thus** *?case*

by (*auto intro*: *terminates.Guard dest*: *notStuck-GuardD*)
next
case *GuardFault* **thus** *?case* **by** (*auto intro*: *terminates.GuardFault*)
next
case *Fault* **show** *?case* **by** (*rule terminates.Fault*)
next
case (*Seq c1 s c2*)
have *not-Stuck*: $\Gamma|_M\vdash_p\langle Seq\ c1\ c2, Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
hence *c1-notStuck*: $\Gamma|_M\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$
by (*rule notStuck-SeqD1*)
show $\Gamma\vdash_p Seq\ c1\ c2 \downarrow Normal\ s$
proof (*rule terminates.Seq,safe*)
from *c1-notStuck*
show $\Gamma\vdash_p c1 \downarrow Normal\ s$
by (*rule Seq.hyps*)
next
fix s'
assume *exec*: $\Gamma\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow s'$
show $\Gamma\vdash_p c2 \downarrow s'$
proof −
from *exec-to-exec-restrict* [*OF exec*] **obtain** t' **where**
exec-res: $\Gamma|_M\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow t'$ **and**
t'-*notStuck*: $t' \neq Stuck \longrightarrow t' = s'$
by *blast*
show *?thesis*
proof (*cases* t'=*Stuck*)
case *True*
with *c1-notStuck exec-res* **have** *False*
by (*auto simp add*: *final-notin-def*)
thus *?thesis* **..**
next
case *False*
with t'-*notStuck* **have** t': t'=s' **by** *simp*
with *not-Stuck exec-res*
have $\Gamma|_M\vdash_p\langle c2, s'\ \rangle \Rightarrow\notin\{Stuck\}$
by (*auto dest*: *notStuck-SeqD2*)
with *exec-res* t' *Seq.hyps*
show *?thesis*
by *auto*
qed
qed
qed
next
case *CondTrue* **thus** *?case*
by (*auto intro*: *terminates.CondTrue dest*: *notStuck-CondTrueD*)
next
case *CondFalse* **thus** *?case*
by (*auto intro*: *terminates.CondFalse dest*: *notStuck-CondFalseD*)
next

```
  case (WhileTrue s b c)
  have s: s ∈ b by fact
  have not-Stuck: Γ|M⊢p⟨While b c,Normal s ⟩ ⇒∉{Stuck} by fact
  with WhileTrue have c-notStuck: Γ|M⊢p⟨c,Normal s ⟩ ⇒∉{Stuck}
    by (iprover intro: notStuck-WhileTrueD1)
  show ?case
  proof (rule terminates.WhileTrue [OF s],safe)
    from c-notStuck
    show Γ⊢p c ↓ Normal s
      by (rule WhileTrue.hyps)
  next
    fix s'
    assume exec: Γ⊢p⟨c,Normal s ⟩ ⇒ s'
    show Γ⊢p While b c ↓ s'
    proof −
      from exec-to-exec-restrict [OF exec] obtain t' where
        exec-res: Γ|M⊢p⟨c,Normal s ⟩ ⇒ t' and
        t'-notStuck: t' ≠ Stuck ⟶ t' = s'
        by blast
      show ?thesis
      proof (cases t'=Stuck)
        case True
        with c-notStuck exec-res have False
          by (auto simp add: final-notin-def)
        thus ?thesis ..
      next
        case False
        with t'-notStuck have t': t'=s' by simp
        with not-Stuck exec-res s
        have Γ|M⊢p⟨While b c,s' ⟩ ⇒∉{Stuck}
          by (auto dest: notStuck-WhileTrueD2)
        with exec-res t' WhileTrue.hyps
        show ?thesis
          by auto
      qed
    qed
  qed
next
  case WhileFalse then show ?case by (iprover intro: terminates.WhileFalse)
next
  case Call thus ?case
    by (auto intro: terminates.Call dest: notStuck-CallD restrict-SomeD)
next
  case CallUndefined
  thus ?case
    by (auto dest: notStuck-CallDefinedD)
next
  case Stuck show ?case by (rule terminates.Stuck)
next
```

case *DynCom*
thus *?case*
  by (*auto intro*: *terminates.DynCom dest*: *notStuck-DynComD*)
next
  case *Throw* **show** *?case* **by** (*rule terminates.Throw*)
next
  case *Abrupt* **show** *?case* **by** (*rule terminates.Abrupt*)
next
  case (*Catch c1 s c2*)
  have *not-Stuck*: $\Gamma|_M\vdash_p\langle Catch\ c1\ c2, Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
  hence *c1-notStuck*: $\Gamma|_M\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$
    by (*rule notStuck-CatchD1*)
  show $\Gamma\vdash_p Catch\ c1\ c2 \downarrow Normal\ s$
  proof (*rule terminates.Catch*,*safe*)
    from *c1-notStuck*
    show $\Gamma\vdash_p c1 \downarrow Normal\ s$
      by (*rule Catch.hyps*)
  next
    fix s'
    assume *exec*: $\Gamma\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
    show $\Gamma\vdash_p c2 \downarrow Normal\ s'$
    proof −
      from *exec-to-exec-restrict* [*OF exec*] **obtain** t' **where**
        exec-res: $\Gamma|_M\vdash_p\langle c1, Normal\ s\ \rangle \Rightarrow t'$ **and**
        t'*-notStuck*: $t' \neq Stuck \longrightarrow t' = Abrupt\ s'$
        by *blast*
      show *?thesis*
      proof (*cases* t'=*Stuck*)
        case *True*
        with *c1-notStuck exec-res* **have** *False*
          by (*auto simp add*: *final-notin-def*)
        thus *?thesis* **..**
      next
        case *False*
        with t'*-notStuck* **have** t': t'=*Abrupt* s' **by** *simp*
        with *not-Stuck exec-res*
        have $\Gamma|_M\vdash_p\langle c2, Normal\ s'\ \rangle \Rightarrow\notin\{Stuck\}$
          by (*auto dest*: *notStuck-CatchD2*)
        with *exec-res* t' *Catch.hyps*
        show *?thesis*
          by *auto*
      qed
    qed
  qed
next
  case (*AwaitTrue s b* Γ_p *c e*)
  then have $(\Gamma|_M)_{\neg a} = (\Gamma_{\neg a})|_M$ **using** *restrict-eq* **by** *auto*
  with *AwaitTrue terminates-restrict-to-terminates* **have** $(\Gamma_{\neg a})|_M\vdash c \downarrow Normal\ s$
   by *force*

then have $\neg\ \Gamma|_M\vdash_p \langle Await\ b\ c\ e, Normal\ s\rangle \Rightarrow Stuck$
by (*fastforce intro*: *AwaitTrue.prems SemanticCon.noStuckE*)
hence $\neg\ \Gamma_{\neg a}|_M\vdash \langle c, Normal\ s\rangle \Rightarrow Stuck$
by (*metis* (*no-types*) *AwaitTrue.hyps*(*1*) ‹$(\Gamma|_M)_{\neg a} = \Gamma_{\neg a}|_M$› *exec.AwaitTrue*)
then have $\Gamma_{\neg a}\vdash c \downarrow Normal\ s$
using *Semantic.noStuckI'* ‹$\Gamma_{\neg a}|_M\vdash c \downarrow Normal\ s$› *terminates-restrict-to-terminates*
by *blast*
thus *?case* **using** *AwaitTrue* **by** (*simp add*: *terminates.AwaitTrue*)
next
case (*AwaitFalse s b*) **thus** *?case* **by** (*simp add*: *terminates.AwaitFalse*)
qed

end

8 Small-Step Semantics and Infinite Computations

theory *SmallStepCon* **imports** *EmbSimpl/SmallStep SemanticCon TerminationCon*

begin

The redex of a statement is the substatement, which is actually altered by the next step in the small-step semantics.

primrec *redex*:: $('s,'p,'f,'e)com \Rightarrow ('s,'p,'f,'e)com$
where
redex Skip = *Skip* |
redex (*Basic f e*) = (*Basic f e*) |
redex (*Spec r e*) = (*Spec r e*) |
redex (*Seq* c_1 c_2) = *redex* c_1 |
redex (*Cond b* c_1 c_2) = (*Cond b* c_1 c_2) |
redex (*While b c*) = (*While b c*) |
redex (*Call p*) = (*Call p*) |
redex (*DynCom d*) = (*DynCom d*) |
redex (*Guard f b c*) = (*Guard f b c*) |
redex (*Throw*) = *Throw* |
redex (*Catch* c_1 c_2) = *redex* c_1 |
redex (*Await b c e*) = (*Await b c e*)

8.1 Small-Step Computation: $\Gamma\vdash_c(c,\ s) \rightarrow (c',\ s')$

type-synonym $('s,'p,'f,'e)$ *config* = $('s,'p,'f,'e)com \times ('s,'f)$ *xstate*

inductive
step-e::[$('s,'p,'f,'e)$ *body*,$('s,'p,'f,'e)$ *config*,$('s,'p,'f,'e)$ *config*] $\Rightarrow$ *bool*
$(\text{-}\vdash_c (\text{-} \rightarrow_e/\ \text{-})\ [81,81,81]\ 100)$
for Γ::$('s,'p,'f,'e)$ *body*
where
Env: $\Gamma\vdash_c (Ps,\ Normal\ s) \rightarrow_e (Ps,\ t)$
|*Env-n*: $(\forall t'.\ t \neq Normal\ t') \Longrightarrow \Gamma\vdash_c (Ps,\ t) \rightarrow_e (Ps,\ t)$

lemma *etranE*: $\Gamma\vdash_c c \rightarrow_e c' \Longrightarrow (\bigwedge P\ s\ t.\ c = (P,\ s) \Longrightarrow c' = (P,\ t) \Longrightarrow Q) \Longrightarrow$
Q
by (*induct c*, *induct c'*, *erule step-e.cases*, *blast*)

inductive-cases *stepe-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_c(Ps,Normal\ s) \rightarrow_e (Ps,t)$

inductive-cases *stepe-elim-cases* [*cases set*]:
$\Gamma\vdash_c(Ps,s) \rightarrow_e (Ps,t)$

inductive-cases *stepe-not-norm-elim-cases* [*cases set*]:
$\Gamma\vdash_c(Ps,s) \rightarrow_e (Ps,Abrupt\ t)$
$\Gamma\vdash_c(Ps,s) \rightarrow_e (Ps,Stuck)$
$\Gamma\vdash_c(Ps,s) \rightarrow_e (Ps,Fault\ t)$
$\Gamma\vdash_c(Ps,s) \rightarrow_e (Ps,Normal\ t)$

lemma *env-c-c'-false*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s')$
shows $^\sim(c=c') \Longrightarrow P$
using *step-m etranE* **by** *blast*

lemma *eenv-normal-s'-normal-s*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ Normal\ s')$
shows $(\bigwedge s1.\ s\neq Normal\ s1) \Longrightarrow P$
using *step-m*
by (*cases*, *auto*)

lemma *env-normal-s'-normal-s*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ Normal\ s')$
shows $\exists s1.\ s= Normal\ s1$
using *step-m*
by (*cases*, *auto*)

lemma *env-c-c'*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s')$
shows $(c=c')$
using *env-c-c'-false step-m* **by** *fastforce*

lemma *env-normal-s*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s') \wedge s\neq s'$
shows $\exists sa.\ s = Normal\ sa$
using *prod.inject step-e.cases step-m* **by** *fastforce*

lemma *env-not-normal-s*:
assumes *a1*:$\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s')$ **and** *a2*:$(\forall t.\ s\neq Normal\ t)$
shows $s=s'$
using *a1 a2*
by (*cases rule*:*step-e.cases*,*auto*)

lemma *env-not-normal-s-not-norma-t*:
  assumes *a1*:$\Gamma\vdash_c$ $(c,\ s) \rightarrow_e (c',\ s')$ **and** *a2*:$(\forall\, t.\ s \neq Normal\ t)$
  shows $(\forall\, t.\ s' \neq Normal\ t)$
using *a1 a2 env-not-normal-s*
by *blast*

lemma *stepe-not-Fault-f-end*:
 assumes *step-e*: $\Gamma\vdash_c$ $(c_1,\ s) \rightarrow_e (c_1',\ s')$
 shows $s' \notin Fault\ `\ f \Longrightarrow s \notin Fault\ `\ f$
proof (*cases s*)
 case (*Fault f'*)
  assume *s'-f*:$s' \notin Fault\ `\ f$ **and**
    $s = Fault\ f'$
  then have $s{=}s'$ **using** *step-e*
  using *env-normal-s xstate.distinct(3)* **by** *blast*
 thus *?thesis* **using** *s'-f Fault* **by** *blast*
qed (*auto*)

inductive
  stepc::[(*'s,'p,'f,'e*) *body*,(*'s,'p,'f,'e*) *config*,(*'s,'p,'f,'e*) *config*] ⇒ *bool*
    (-$\vdash_c$ (- →/ -) [*81,81,81*] *100*)
 for Γ::(*'s,'p,'f,'e*) *body*
where

 Basicc: $\Gamma\vdash_c(Basic\ f\ e, Normal\ s) \rightarrow (Skip, Normal\ (f\ s))$

| *Specc*: $(s,t) \in r \Longrightarrow \Gamma\vdash_c(Spec\ r\ e, Normal\ s) \rightarrow (Skip, Normal\ t)$
| *SpecStuckc*: $\forall\, t.\ (s,t) \notin r \Longrightarrow \Gamma\vdash_c(Spec\ r\ e, Normal\ s) \rightarrow (Skip, Stuck)$

| *Guardc*: $s{\in}g \Longrightarrow \Gamma\vdash_c(Guard\ f\ g\ c, Normal\ s) \rightarrow (c, Normal\ s)$

| *GuardFaultc*: $s{\notin}g \Longrightarrow \Gamma\vdash_c(Guard\ f\ g\ c, Normal\ s) \rightarrow (Skip, Fault\ f)$

| *Seqc*: $\Gamma\vdash_c(c_1,s) \rightarrow (c_1',s')$
  $\Longrightarrow$
  $\Gamma\vdash_c(Seq\ c_1\ c_2,s) \rightarrow (Seq\ c_1'\ c_2,\ s')$
| *SeqSkipc*: $\Gamma\vdash_c(Seq\ Skip\ c_2,s) \rightarrow (c_2,\ s)$
| *SeqThrowc*: $\Gamma\vdash_c(Seq\ Throw\ c_2, Normal\ s) \rightarrow (Throw,\ Normal\ s)$

| *CondTruec*: $s{\in}b \Longrightarrow \Gamma\vdash_c(Cond\ b\ c_1\ c_2, Normal\ s) \rightarrow (c_1, Normal\ s)$
| *CondFalsec*: $s{\notin}b \Longrightarrow \Gamma\vdash_c(Cond\ b\ c_1\ c_2, Normal\ s) \rightarrow (c_2, Normal\ s)$

| *WhileTruec*: $[\![s{\in}b]\!]$
    $\Longrightarrow$
    $\Gamma\vdash_c(While\ b\ c, Normal\ s) \rightarrow (Seq\ c\ (While\ b\ c), Normal\ s)$

| *WhileFalsec*: $[\![s{\notin}b]\!]$

$\Longrightarrow$
$\Gamma\vdash_c(\mathit{While}\ b\ c,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Normal}\ s)$

| *Awaitc*: $[\![s{\in}b;\ \Gamma 1{=}\Gamma_{\neg a}\ ;\ \Gamma 1\vdash\langle ca1,\mathit{Normal}\ s\rangle \Rightarrow t;$
$\neg(\exists\, t'.\ t = \mathit{Abrupt}\ t')]\!] \Longrightarrow$
$\Gamma\vdash_c(\mathit{Await}\ b\ ca1\ e,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},t)$

| *AwaitAbruptc*: $[\![s{\in}b;\ \Gamma 1{=}\Gamma_{\neg a}\ ;\ \Gamma 1\vdash\langle ca1,\mathit{Normal}\ s\rangle \Rightarrow t;$
$t = \mathit{Abrupt}\ t']\!] \Longrightarrow$
$\Gamma\vdash_c(\mathit{Await}\ b\ ca1\ e,\mathit{Normal}\ s) \rightarrow (\mathit{Throw},\mathit{Normal}\ t')$

| *Callc*: $[\![\Gamma\ p = \mathit{Some}\ \mathit{bdy}\ ;\ \mathit{bdy}{\neq}\mathit{Call}\ p]\!] \Longrightarrow$
$\Gamma\vdash_c(\mathit{Call}\ p,\mathit{Normal}\ s) \rightarrow (\mathit{bdy},\mathit{Normal}\ s)$

| *CallUndefinedc*: $\Gamma\ p{=}\mathit{None} \Longrightarrow$
$\Gamma\vdash_c(\mathit{Call}\ p,\mathit{Normal}\ s) \rightarrow (\mathit{Skip},\mathit{Stuck})$

| *DynComc*: $\Gamma\vdash_c(\mathit{DynCom}\ c,\mathit{Normal}\ s) \rightarrow (c\ s,\mathit{Normal}\ s)$

| *Catchc*: $[\![\Gamma\vdash_c(c_1,s) \rightarrow (c_1{}',s')]\!]$
$\Longrightarrow$
$\Gamma\vdash_c(\mathit{Catch}\ c_1\ c_2,s) \rightarrow (\mathit{Catch}\ c_1{}'\ c_2,s')$

| *CatchThrowc*: $\Gamma\vdash_c(\mathit{Catch}\ \mathit{Throw}\ c_2,\mathit{Normal}\ s) \rightarrow (c_2,\mathit{Normal}\ s)$
| *CatchSkipc*: $\Gamma\vdash_c(\mathit{Catch}\ \mathit{Skip}\ c_2,s) \rightarrow (\mathit{Skip},s)$

| *FaultPropc*: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash_c(c,\mathit{Fault}\ f) \rightarrow (\mathit{Skip},\mathit{Fault}\ f)$
| *StuckPropc*: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash_c(c,\mathit{Stuck}) \rightarrow (\mathit{Skip},\mathit{Stuck})$
| *AbruptPropc*: $[\![c{\neq}\mathit{Skip};\ \mathit{redex}\ c = c]\!] \Longrightarrow \Gamma\vdash_c(c,\mathit{Abrupt}\ f) \rightarrow (\mathit{Skip},\mathit{Abrupt}\ f)$

lemmas *stepc-induct* = *stepc.induct* [*of* - (c,s) (c',s'), *split-format* (*complete*), *case-names*
Basicc Specc SpecStuckc Guardc GuardFaultc Seqc SeqSkipc SeqThrowc CondTruec CondFalsec
WhileTruec WhileFalsec Awaitc AwaitAbruptc Callc CallUndefinedc DynComc Catchc CatchThrowc CatchSkipc
FaultPropc StuckPropc AbruptPropc, *induct set*]

inductive-cases *stepc-elim-cases* [*cases set*]:
$\Gamma\vdash_c(\mathit{Skip},s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Guard}\ f\ g\ c,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Basic}\ f\ e,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Spec}\ r\ e,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Seq}\ c1\ c2,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Cond}\ b\ c1\ c2,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{While}\ b\ c,s) \rightarrow u$
$\Gamma\vdash_c(\mathit{Await}\ b\ c2\ e,s) \rightarrow u$

$\Gamma\vdash_c(Call\ p,s) \to u$
$\Gamma\vdash_c(DynCom\ c,s) \to u$
$\Gamma\vdash_c(Throw,s) \to u$
$\Gamma\vdash_c(Catch\ c1\ c2,s) \to u$

inductive-cases *stepc-not-normal-elim-cases*:
$\Gamma\vdash_c(Call\ p,Abrupt\ s) \to (p',s')$
$\Gamma\vdash_c(Call\ p,\ Fault\ f) \to (p',s')$
$\Gamma\vdash_c(Call\ p,\ Stuck) \to (p',s')$

lemma *Guardc-not-c*:*Guard f g c* $\neq$ *c*
proof (*induct c*)
qed *auto*

lemma *Catch-not-c1*:*Catch c1 c2* $\neq$ *c1*
proof (*induct c1*)
qed *auto*

lemma *Catch-not-c*:*Catch c1 c2* $\neq$ *c2*
proof (*induct c2*)
qed *auto*

lemma *seq-not-eq1*: *Seq c1 c2*$\neq$*c1*
 by (*induct c1*) *auto*

lemma *seq-not-eq2*: *Seq c1 c2*$\neq$*c2*
 by (*induct c2*) *auto*

lemma *if-not-eq1*: *Cond b c1 c2* $\neq$*c1*
 by (*induct c1*) *auto*

lemma *if-not-eq2*: *Cond b c1 c2*$\neq$*c2*
 by (*induct c2*) *auto*

lemmas *seq-and-if-not-eq* [*simp*] = *seq-not-eq1 seq-not-eq2*
seq-not-eq1 [*THEN not-sym*] *seq-not-eq2* [*THEN not-sym*]
if-not-eq1 if-not-eq2 if-not-eq1 [*THEN not-sym*] *if-not-eq2* [*THEN not-sym*]
Catch-not-c1 Catch-not-c Catch-not-c1 [*THEN not-sym*] *Catch-not-c*[*THEN not-sym*]

Guardc-not-c Guardc-not-c[*THEN not-sym*]

inductive-cases *stepc-elim-cases-Seq-Seq*:
$\Gamma\vdash_c(Seq\ c1\ c2,s) \to (Seq\ c1'\ c2,s')$

inductive-cases *stepc-elim-cases-Seq-Seq1*:
$\Gamma\vdash_c(Seq\ c1\ c2,Fault\ f) \to (q,s')$
thm *stepc-elim-cases-Seq-Seq1*

inductive-cases *stepc-elim-cases-Catch-Catch*:
$\Gamma \vdash_c (Catch\ c1\ c2,s) \rightarrow (Catch\ c1'\ c2,s')$

inductive-cases *stepc-elim-cases-Catch-Catch1*:
$\Gamma \vdash_c (Seq\ c1\ c2,Fault\ f) \rightarrow (q,s')$

inductive-cases *stepc-elim-cases-Seq-skip*:
$\Gamma \vdash_c (Seq\ Skip\ c2,s) \rightarrow u$
$\Gamma \vdash_c (Seq\ (Guard\ f\ g\ c1)\ c2,s) \rightarrow u$

inductive-cases *stepc-elim-cases-Catch-skip*:
$\Gamma \vdash_c (Catch\ Skip\ c2,s) \rightarrow u$

inductive-cases *stepc-elim-cases-Await-skip*:
$\Gamma \vdash_c (Await\ b\ c\ e,\ Normal\ s) \rightarrow (Skip,t)$

inductive-cases *stepc-elim-cases-Await-throw*:
$\Gamma \vdash_c (Await\ b\ c\ e,\ Normal\ s) \rightarrow (Throw,t)$

inductive-cases *stepc-elim-cases-Catch-throw*:
$\Gamma \vdash_c (Catch\ c1\ c2,s) \rightarrow (Throw,\ Normal\ s1)$

inductive-cases *stepc-elim-cases-Catch-skip-c2*:
$\Gamma \vdash_c (Catch\ c1\ c2,s) \rightarrow (c2,s)$

inductive-cases *stepc-Normal-elim-cases* [*cases set*]:
$\Gamma \vdash_c (Skip,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Guard\ f\ g\ c,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Basic\ f\ e,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Spec\ r\ e,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Seq\ c1\ c2,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Cond\ b\ c1\ c2,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (While\ b\ c,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Await\ b\ c\ e,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Call\ p,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (DynCom\ c,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Throw,Normal\ s) \rightarrow u$
$\Gamma \vdash_c (Catch\ c1\ c2,Normal\ s) \rightarrow u$

The final configuration is either of the form (*Skip*,-) for normal termination, or (*LanguageCon.com.Throw*, *Normal s*) in case the program was started in a *Normal* state and terminated abruptly. The *Abrupt* state is not used to model abrupt termination, in contrast to the big-step semantics. Only if the program starts in an *Abrupt* states it ends in the same *Abrupt* state.

definition *final*:: $('s,'p,'f,'e)$ *config* $\Rightarrow$ *bool* **where**
final cfg $\equiv$ (*fst cfg*=*Skip* $\vee$ ((*fst cfg*=*Throw*) $\wedge$ ($\exists$ *s*. *snd cfg*=*Normal s*)))

definition *final-valid*::$('s,'p,'f,'e)$ *config* $\Rightarrow$ *bool* **where**
final-valid cfg = ((*fst cfg*=*Skip* $\vee$ *fst cfg*=*Throw*) $\wedge$ ($\exists$ *s*. *snd cfg*=*Normal s*))

abbreviation
stepc-rtrancl :: $[('s,'p,'f,'e)$ *body*,$('s,'p,'f,'e)$ *config*,$('s,'p,'f,'e)$ *config*$]$ $\Rightarrow$ *bool*
($\text{-}\vdash_c$ ($\text{-} \rightarrow^*/ \text{-}$) [*81*,*81*,*81*] *100*)
where
$\Gamma\vdash_c$ *cf0* $\rightarrow^*$ *cf1* $\equiv$ ((*CONST stepc* Γ))** *cf0 cf1*

abbreviation
stepc-trancl :: $[('s,'p,'f,'e)$ *body*,$('s,'p,'f,'e)$ *config*,$('s,'p,'f,'e)$ *config*$]$ $\Rightarrow$ *bool*
($\text{-}\vdash_c$ ($\text{-} \rightarrow^+/ \text{-}$) [*81*,*81*,*81*] *100*)
where
$\Gamma\vdash_c$ *cf0* $\rightarrow^+$ *cf1* $\equiv$ (*CONST stepc* Γ)$^{++}$ *cf0 cf1*

lemma
assumes
step-a: $\Gamma\vdash_c$(*Await b c e*, *Normal s*) $\rightarrow$ (*t*,*u*)
shows *step-await-step-c*:($\Gamma_{\neg a}$)$\vdash$(*c*, *Normal s*) $\rightarrow^*$ (*sequential t*,*u*)
using *step-a*
proof *cases*
fix *t1*
assume
(*t*, *u*) = (*Skip*, *t1*) *s* $\in$ *b* ($\Gamma_{\neg a}$)$\vdash$ $\langle$*c*,*Normal s*$\rangle$ $\Rightarrow$ *t1* $\forall$ *t'*. *t1* $\neq$ *Abrupt t'*
thus *?thesis*
by (*cases u*)
(*auto intro*: *exec-impl-steps-Fault exec-impl-steps-Normal exec-impl-steps-Stuck*)
next
fix *t1*
assume (*t*, *u*) = (*Throw*, *Normal t1*) *s* $\in$ *b* ($\Gamma_{\neg a}$)$\vdash$ $\langle$*c*,*Normal s*$\rangle$ $\Rightarrow$ *Abrupt t1*
thus *?thesis* **by** (*simp add*: *exec-impl-steps-Normal-Abrupt*)
qed

lemma
assumes
step-a: $\Gamma\vdash_c$(*Await b c e*, *Normal s*) $\rightarrow$ *u*
shows *step-await-final1*:*final u*
using *step-a*
proof *cases*
case (*1 t*) **thus** *final u* **by** (*simp add*: *final-def*)
next
case (*2 t*)
thus *final u* **by** (*simp add*: *exec-impl-steps-Normal-Abrupt final-def*)
qed

lemma *step-Abrupt-end*:
assumes *step*: $\Gamma\vdash_c$ (c_1, *s*) $\rightarrow$ (c_1', *s'*)
shows *s'*=*Abrupt x* $\Longrightarrow$ *s*=*Abrupt x*

using *step*
by *induct auto*

lemma *step-Stuck-end*:
assumes *step*: $\Gamma\vdash_c (c_1, s) \to (c_1', s')$
shows s'=*Stuck* $\Longrightarrow$
s=*Stuck* $\vee$
$(\exists\, r\ x\ e.\ redex\ c_1 = Spec\ r\ e \wedge s{=}Normal\ x \wedge (\forall\, t.\ (x,t)\notin r)) \vee$
$(\exists\, p\ x.\ redex\ c_1 = \ Call\ p \wedge s{=}Normal\ x \wedge \Gamma\ p = None) \vee$
$(\exists\, b\ c\ x\ e.\ \ redex\ c_1 = Await\ b\ c\ e \wedge s{=}Normal\ x \wedge x \in b \wedge (\Gamma_{\neg a})\vdash\langle c,s\rangle\Rightarrow s')$
using *step*
by *induct auto*

lemma *step-Fault-end*:
assumes *step*: $\Gamma\vdash_c (c_1, s) \to (c_1', s')$
shows s'=*Fault f* $\Longrightarrow$
s=*Fault f* $\vee$
$(\exists\, g\ c\ x.\ \ redex\ c_1 = Guard\ f\ g\ c \wedge s{=}Normal\ x \wedge x \notin g) \vee$
$(\exists\, b\ \ c1\ \ x\ \ e.\ \ \ redex\ c_1 = Await\ b\ c1\ e \wedge s{=}Normal\ x \wedge x \in b \wedge (\Gamma_{\neg a})\vdash\langle c1,s\rangle\Rightarrow s')$
using *step*
by *induct auto*

lemma *step-not-Fault-f-end*:
assumes *step*: $\Gamma\vdash_c (c_1, s) \to (c_1', s')$
shows $s'\notin$ *Fault ‘ f* $\Longrightarrow$ $s \notin$ *Fault ‘ f*
using *step*
by *induct auto*

inductive
step-ce::[('s,'p,'f,'e) *body*,('s,'p,'f,'e) *config*,('s,'p,'f,'e) *config*] $\Rightarrow$ *bool*
$(\text{-}\vdash_c (\text{-} \to_{ce}/ \text{-})\ [81,81,81]\ 100)$
for Γ::('s,'p,'f,'e) *body*
where
c-step: $\Gamma\vdash_c cf0 \to cf1 \Longrightarrow \Gamma\vdash_c cf0 \to_{ce} cf1$
|*e-step*: $\Gamma\vdash_c cf0 \to_e cf1 \Longrightarrow \Gamma\vdash_c cf0 \to_{ce} cf1$

lemmas *step-ce-induct* = *step-ce.induct* [*of* - (c,s) (c',s'), *split-format* (*complete*), *case-names*
c-step e-step, *induct set*]

inductive-cases *step-ce-elim-cases* [*cases set*]:
$\Gamma\vdash_c cf0 \to_{ce} cf1$

lemma *step-c-normal-normal*: **assumes** *a1*: $\Gamma\vdash_c$ *cf0* $\rightarrow$ *cf1*
shows $\bigwedge$ c_1 *s s'*. ⟦*cf0* = (c_1,*Normal s*);*cf1*=(c_1,*s'*);(∀ *sa*. ¬(*s'*=*Normal sa*))⟧
⟹ *P*
using *a1*
by (*induct rule*: *stepc.induct*, *induct*, *auto*)

lemma *normal-not-normal-eq-p*:
assumes *a1*: $\Gamma\vdash_c$ *cf0* $\rightarrow_{ce}$ *cf1*
shows $\bigwedge$ c_1 *s s'*. ⟦*cf0* = (c_1,*Normal s*);*cf1*=(c_1,*s'*);(∀ *sa*. ¬(*s'*=*Normal sa*))⟧
⟹ $\Gamma\vdash_c$ *cf0* $\rightarrow_e$ *cf1* ∧ ¬($\Gamma\vdash_c$ *cf0* $\rightarrow$ *cf1*)
by (*meson step-c-normal-normal step-e.intros*)

lemma *call-not-normal-skip-always*:
assumes *a0*:$\Gamma\vdash_c$(*Call p*,*s*) $\rightarrow$ (*p1*,*s1*) **and**
a1:∀ *sn*. *s* ≠ *Normal sn* **and**
a2:*p1*≠*Skip*
shows *P*
proof(*cases s*)
case *Normal* **thus** *?thesis* **using** *a1* **by** *fastforce*
next
case *Stuck*
then have *a0*:$\Gamma\vdash_c$(*Call p*,*Stuck*) $\rightarrow$ (*p1*,*s1*) **using** *a0* **by** *auto*
show *?thesis* **using** *a1 a2 stepc-not-normal-elim-cases*(*3*)[*OF a0*] **by** *fastforce*
next
case (*Fault f*)
then have *a0*:$\Gamma\vdash_c$(*Call p*,*Fault f*) $\rightarrow$ (*p1*,*s1*) **using** *a0* **by** *auto*
show *?thesis* **using** *a1 a2 stepc-not-normal-elim-cases*(*2*)[*OF a0*] **by** *fastforce*
next
case (*Abrupt a*)
then have *a0*:$\Gamma\vdash_c$(*Call p*,*Abrupt a*) $\rightarrow$ (*p1*,*s1*) **using** *a0* **by** *auto*
show *?thesis* **using** *a1 a2 stepc-not-normal-elim-cases*(*1*)[*OF a0*] **by** *fastforce*
qed

lemma *call-f-step-not-s-eq-t-false*:
assumes
a0:$\Gamma\vdash_c$(*P*,*s*) $\rightarrow$ (*Q*,*t*) **and**
a1:(*redex P* = *Call fn* ∧ Γ *fn* = *Some bdy* ∧ *s*=*Normal s'* ∧ ~(*s*=*t*)) ∨
(*redex P* = *Call fn* ∧ Γ *fn* = *Some bdy* ∧ *s*=*Normal s'* ∧ *s*=*t* ∧ *P*=*Q* ∧ Γ
fn ≠ *Some* (*Call fn*))
shows *False*
using *a0 a1*
proof (*induct rule*:*stepc-induct*)
qed(*fastforce+*,*auto*)

lemma *call-f-step-ce-not-s-eq-t-env-step*:
assumes
a0:$\Gamma\vdash_c$(*P*,*s*) $\rightarrow_{ce}$ (*Q*,*t*) **and**
a1:(*redex P* = *Call fn* ∧ Γ *fn* = *Some bdy* ∧ *s*=*Normal s'* ∧ ~(*s*=*t*)) ∨
(*redex P* = *Call fn* ∧ Γ *fn* = *Some bdy* ∧ *s*=*Normal s'* ∧ *s*=*t* ∧ *P*=*Q* ∧ Γ

$fn \neq Some\ (Call\ fn))$
shows $\Gamma\vdash_c(P,s) \rightarrow_e (Q,t)$
proof−
have $\Gamma\vdash_c(P,s) \rightarrow_e (Q,t) \vee \Gamma\vdash_c(P,s) \rightarrow (Q,t)$
using *a0 step-ce-elim-cases* **by** *fastforce*
thus *?thesis* **using** *call-f-step-not-s-eq-t-false a1* **by** *fastforce*
qed

abbreviation
stepce-rtrancl :: $[('s,'p,'f,'e)\ body,('s,'p,'f,'e)\ config,('s,'p,'f,'e)\ config] \Rightarrow bool$
$(\text{-}\vdash_c\ (\text{-} \rightarrow_{ce}{}^*/\ \text{-})\ [81,81,81]\ 100)$
where
$\Gamma\vdash_c\ cf0 \rightarrow_{ce}{}^*\ cf1 \equiv ((CONST\ step\text{-}ce\ \Gamma))^{**}\ cf0\ cf1$

abbreviation
stepce-trancl :: $[('s,'p,'f,'e)\ body,('s,'p,'f,'e)\ config,('s,'p,'f,'e)\ config] \Rightarrow bool$
$(\text{-}\vdash_c\ (\text{-} \rightarrow_{ce}{}^+/\ \text{-})\ [81,81,81]\ 100)$
where
$\Gamma\vdash_c\ cf0 \rightarrow_{ce}{}^+\ cf1 \equiv (CONST\ step\text{-}ce\ \Gamma)^{++}\ cf0\ cf1$

8.2 Parallel Computation: $\Gamma\vdash(c,\ s) \rightarrow_p\ (c',\ s')$

type-synonym $('s,'p,'f,'e)$ *par-Simpl* $= ('s,'p,'f,'e)com\ \ list$
type-synonym $('s,'p,'f,'e)$ *par-config* $= ('s,'p,'f,'e)$ *par-Simpl* $\times ('s,'f)$ *xstate*

definition *final-c*:: $('s,'p,'f,'e)$ *par-config* $\Rightarrow$ *bool* **where**
final-c cfg $= (\forall i.\ i<length\ (fst\ cfg) \longrightarrow final\ ((fst\ cfg)!i,\ snd\ cfg))$

inductive
step-pe:: $[('s,'p,'f,'e)\ body,('s,'p,'f,'e)\ par\text{-}config,('s,'p,'f,'e)\ par\text{-}config] \Rightarrow bool$
$(\text{-}\vdash_p\ (\text{-} \rightarrow_e/\ \text{-})\ [81,81,81]\ 100)$
for Γ::$('s,'p,'f,'e)$ *body*
where
ParEnv: $\Gamma\vdash_p\ (Ps,\ Normal\ s) \rightarrow_e (Ps,\ Normal\ t)$

lemma *ptranE*: $\Gamma\vdash_p\ c \rightarrow_e c' \Longrightarrow (\bigwedge P\ s\ t.\ c = (P,\ s) \Longrightarrow c' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
by (*induct c*, *induct c'*, *erule step-pe.cases*, *blast*)

inductive-cases *step-pe-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_p(PS,Normal\ s) \rightarrow_e (Ps,t)$

inductive-cases *step-pe-elim-cases* [*cases set*]:

$\Gamma\vdash_p(PS,s) \rightarrow_e (Ps,t)$

inductive-cases *step-pe-not-norm-elim-cases* [*cases set*]:
$\Gamma\vdash_p(Ps,s) \rightarrow_e (Ps,Abrupt\ t)$
$\Gamma\vdash_p(Ps,s) \rightarrow_e (Ps,Stuck)$
$\Gamma\vdash_p(Ps,s) \rightarrow_e (Ps,Fault\ t)$

lemma *env-pe-c-c′-false*:
  assumes *step-m*: $\Gamma\vdash_p (c,\ s) \rightarrow_e (c',\ s')$
  shows $\sim(c=c') \Longrightarrow P$
using *step-m ptranE* **by** *blast*

lemma *env-pe-c-c′*:
  assumes *step-m*: $\Gamma\vdash_p (c,\ s) \rightarrow_e (c',\ s')$
  shows $(c=c')$
using *env-pe-c-c′-false step-m* **by** *fastforce*

lemma *env-pe-normal-s*:
  assumes *step-m*: $\Gamma\vdash_p (c,\ s) \rightarrow_e (c',\ s') \wedge s\neq s'$
  shows $\exists\, sa.\ s = Normal\ sa$
using *prod.inject step-pe.cases step-m* **by** *fastforce*

lemma *env-pe-not-normal-s*:
  assumes *a1*:$\Gamma\vdash_p (c,\ s) \rightarrow_e (c',\ s')$ **and** *a2*:$(\forall\, t.\ s\neq Normal\ t)$
  shows $s=s'$
using *a1 a2*
by (*cases rule*:*step-pe.cases*,*auto*)

lemma *env-pe-not-normal-s-not-norma-t*:
  assumes *a1*:$\Gamma\vdash_p (c,\ s) \rightarrow_e (c',\ s')$ **and** *a2*:$(\forall\, t.\ s\neq Normal\ t)$
  shows $(\forall\, t.\ s'\neq Normal\ t)$
using *a1 a2 env-pe-not-normal-s*
by *blast*

inductive
step-p::[(′*s*,′*p*,′*f*,′*e*) *body*, (′*s*,′*p*,′*f*,′*e*) *par-config*,
    (′*s*,′*p*,′*f*,′*e*) *par-config*] ⇒ *bool*
$(\text{-}\vdash_p (\text{-} \rightarrow/\ \text{-})\ [81,81,81]\ 100)$
where
ParComp: $[\![i<length\ Ps;\ \ \Gamma\vdash_c(Ps!i,s) \rightarrow (r,s')]\!] \Longrightarrow$
    $\Gamma\vdash_p(Ps,\ s) \rightarrow (Ps[i:=r],\ s')$

lemmas *steppe-induct* = *step-p.induct* [*of* - (*c*,*s*) (*c*′,*s*′), *split-format* (*complete*), *case-names*
ParComp, *induct set*]

inductive-cases *step-p-elim-cases* [*cases set*]:
$\Gamma\vdash_p(Ps,\ s) \rightarrow u$

inductive-cases *step-p-pair-elim-cases* [*cases set*]:
$\Gamma\vdash_p(Ps,\ s) \rightarrow (Qs,\ t)$

inductive-cases *step-p-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash_p(Ps,\ Normal\ s) \rightarrow u$

lemma *par-ctranE*: $\Gamma \vdash_p c \rightarrow c' \Longrightarrow$
$(\bigwedge i\ Ps\ s\ r\ t.\ c = (Ps,\ s) \Longrightarrow c' = (Ps[i := r],\ t) \Longrightarrow i < length\ Ps \Longrightarrow$
$\Gamma \vdash_c (Ps!i,\ s) \rightarrow (r,\ t) \Longrightarrow P) \Longrightarrow P$
by (*induct c*, *induct c'*, *erule step-p.cases*, *blast*)

8.3 Computations

8.3.1 Sequential computations

type-synonym $('s,'p,'f,'e)$ *confs* =
$('s,'p,'f,'e)$ *body* $\times(('s,'p,'f,'e)$ *config*) *list*

inductive-set *cptn* :: $(('s,'p,'f,'e)$ *confs*) *set*
where
CptnOne: $(\Gamma,\ [(P,s)]) \in cptn$
| *CptnEnv*: $[\![\Gamma\vdash_c(P,s) \rightarrow_e (P,t);\ (\Gamma,(P,\ t)\#xs) \in cptn\ ]\!] \Longrightarrow$
$(\Gamma,(P,s)\#(P,t)\#xs) \in cptn$
| *CptnComp*: $[\![\Gamma\vdash_c(P,s) \rightarrow (Q,t);\ (\Gamma,(Q,\ t)\#xs) \in cptn\ ]\!] \Longrightarrow$
$(\Gamma,(P,s)\#(Q,t)\#xs) \in cptn$

inductive-cases *cptn-elim-cases* [*cases set*]:
$(\Gamma,\ [(P,s)]) \in cptn$
$(\Gamma,(P,s)\#(Q,t)\#xs) \in cptn$
$(\Gamma,(P,s)\#(P,t)\#xs) \in cptn$

inductive-cases *cptn-elim-cases-pair* [*cases set*]:
$(\Gamma,\ [x]) \in cptn$
$(\Gamma,\ x\#x1\#xs) \in cptn$

lemma *cptn-dest*:$(\Gamma,(P,s)\#(Q,t)\#xs) \in cptn \Longrightarrow (\Gamma,(Q,t)\#xs)\in cptn$
by (*auto dest*: *cptn-elim-cases*)

lemma *cptn-dest-pair*:$(\Gamma,x\#x1\#xs) \in cptn \Longrightarrow (\Gamma,x1\#xs)\in cptn$
proof −
assume $(\Gamma,x\#x1\#xs) \in cptn$
thus *?thesis* **using** *cptn-dest prod.collapse* **by** *metis*
qed

lemma *cptn-dest1*:$(\Gamma,(P,s)\#(Q,t)\#xs) \in cptn \Longrightarrow (\Gamma,(P,s)\#[(Q,t)])\in cptn$
proof −
assume *a1*: $(\Gamma,\ (P,\ s)\ \#\ (Q,\ t)\ \#\ xs) \in cptn$
have $(\Gamma,\ [(Q,\ t)]) \in cptn$

by (*meson cptn.CptnOne*)
thus *?thesis*
proof (*cases s*)
case (*Normal s'*)
then have *f1*: $(\Gamma, (P, Normal\ s') \# (Q, t) \# xs) \in cptn$
using *Normal a1* **by** *blast*
have $(\Gamma, [(P, t)]) \in cptn \longrightarrow (\Gamma, [(P, Normal\ s'), (P, t)]) \in cptn$
by (*simp add*: *Env cptn.CptnEnv*)
thus *?thesis*
using *f1* **by** (*metis* (*no-types*) *Normal* ‹$(\Gamma, [(Q, t)]) \in cptn$› *cptn.CptnComp cptn-elim-cases*(*2*))
next
case (*Abrupt x*) **thus** *?thesis*
using ‹$(\Gamma, [(Q, t)]) \in cptn$› *a1 cptn.CptnComp cptn-elim-cases*(*2*) *CptnEnv* **by** *metis*
next
case (*Stuck*) **thus** *?thesis*
using ‹$(\Gamma, [(Q, t)]) \in cptn$› *a1 cptn.CptnComp cptn-elim-cases*(*2*) *CptnEnv* **by** *metis*
next
case (*Fault f*) **thus** *?thesis*
using ‹$(\Gamma, [(Q, t)]) \in cptn$› *a1 cptn.CptnComp cptn-elim-cases*(*2*) *CptnEnv* **by** *metis*
qed
qed

lemma *cptn-dest1-pair*:$(\Gamma,x\#x1\#xs) \in cptn \Longrightarrow (\Gamma,x\#[x1])\in cptn$
proof −
assume $(\Gamma,x\#x1\#xs) \in cptn$
thus *?thesis* **using** *cptn-dest1 prod.collapse* **by** *metis*
qed

lemma *cptn-append-is-cptn* [*rule-format*]:
$\forall b\ a. (\Gamma,b\#c1)\in cptn \longrightarrow (\Gamma,a\#c2)\in cptn \longrightarrow (b\#c1)!length\ c1=a \longrightarrow (\Gamma,b\#c1@c2)\in cptn$
apply(*induct c1*)
apply *simp*
apply *clarify*
apply(*erule cptn.cases*,*simp-all*)
apply (*simp add*: *cptn.CptnEnv*)
by (*simp add*: *cptn.CptnComp*)

lemma *cptn-dest-2*:
$(\Gamma,a\#xs@ys) \in cptn \Longrightarrow (\Gamma,a\#xs)\in cptn$
proof (*induct xs arbitrary*: *a*)
case *Nil* **thus** *?case* **using** *cptn.simps* **by** *fastforce*
next
case (*Cons x xs'*)
then have $(\Gamma,a\#[x])\in cptn$ **by** (*simp add*: *cptn-dest1-pair*)
also have $(\Gamma, x \# xs') \in cptn$

using *Cons.hyps Cons.prems cptn-dest-pair* **by** *fastforce*
ultimately show *?case* **using** *cptn-append-is-cptn* [*of* Γ *a* [*x*] *x xs'*]
by *force*
qed

lemma *last-not-F*:
assumes
a0:(Γ,*xs*)∈*cptn*
shows *snd* (*last xs*) ∉ *Fault* ' *F* ⟹ ∀ *i* < *length xs*. *snd* (*xs*!*i*) ∉ *Fault* ' *F*
using *a0*
proof(*induct*) **print-cases**
case (*CptnOne* Γ *p s*) **thus** *?case* **by** *auto*
next
case (*CptnEnv* Γ *P s t xs*)
thus *?case* **using** *stepe-not-Fault-f-end*
proof −
{ **fix** *nn* :: *nat*
have *snd* (*last* ((*P*, *t*) # *xs*)) ∉ *Fault* ' *F*
using *CptnEnv.prems* **by** *force*
then have ¬ *nn* < *length* ((*P*, *s*) # (*P*, *t*) # *xs*) ∨ *snd* (((*P*, *s*) # (*P*, *t*) # *xs*) ! *nn*) ∉ *Fault* ' *F*
by (*metis* (*no-types*) *CptnEnv.hyps*(*1*) *CptnEnv.hyps*(*3*) *length-Cons less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc snd-conv stepe-not-Fault-f-end*)
}
then have ∀ *n*. ¬ *n* < *length* ((*P*, *s*) # (*P*, *t*) # *xs*) ∨ *snd* (((*P*, *s*) # (*P*, *t*) # *xs*) ! *n*) ∉ *Fault* ' *F*
by *meson*
then show *?thesis*
by *metis*
qed
next
case (*CptnComp* Γ *P s Q t xs*)
have *snd* (*last* ((*Q*, *t*) # *xs*)) ∉ *Fault* ' *F*
using *CptnComp.prems* **by** *force*
then have *all*:∀ *i*<*length* ((*Q*, *t*) # *xs*). *snd* (((*Q*, *t*) # *xs*) ! *i*) ∉ *Fault* ' *F*
using *CptnComp.hyps* **by** *force*
then have *t* ∉ *Fault* ' *F*
by *force*
then have *s* ∉ *Fault* ' *F* **using** *step-not-Fault-f-end*
using *CptnComp.hyps*(*1*) **by** *blast*
then have *zero*:*snd* (*P*,*s*) ∉ *Fault* ' *F* **by** *auto*
show *?case*
proof −
{ **fix** *nn* :: *nat*
have ¬ *nn* < *length* ((*P*, *s*) # (*Q*, *t*) # *xs*) ∨ *snd* (((*P*, *s*) # (*Q*, *t*) # *xs*) ! *nn*) ∉ *Fault* ' *F*
by (*metis* (*no-types*) ‹∀ *i*<*length* ((*Q*, *t*) # *xs*). *snd* (((*Q*, *t*) # *xs*) ! *i*) ∉ *Fault* ' *F*› ‹*snd* (*P*, *s*) ∉ *Fault* ' *F*› *diff-Suc-1 length-Cons less-Suc-eq-0-disj nth-Cons'*)

}
then show *?thesis*
by *meson*
qed
qed

definition *cp* :: $('s,'p,'f,'e)$ *body* $\Rightarrow$ $('s,'p,'f,'e)$ *com* $\Rightarrow$
$('s,'f)$ *xstate* $\Rightarrow$ $(('s,'p,'f,'e)$ *confs*$)$ *set* **where**
cp Γ P $s \equiv \{(\Gamma 1,l).\ l!0=(P,s) \wedge (\Gamma,l) \in cptn \wedge \Gamma 1=\Gamma\}$

lemma *cp-sub*:
assumes *a0*: $(\Gamma,(x\#l0)@l1) \in cp\ \Gamma\ P\ s$
shows $(\Gamma,(x\#l0)) \in cp\ \Gamma\ P\ s$
proof −
have $(x\#l0)!0 = (P,s)$ **using** *a0* **unfolding** *cp-def* **by** *auto*
also have $(\Gamma,(x\#l0)) \in cptn$ **using** *a0* **unfolding** *cp-def*
using *cptn-dest-2* **by** *fastforce*
ultimately show *?thesis* **using** *a0* **unfolding** *cp-def* **by** *blast*
qed

8.3.2 Parallel computations

type-synonym $('s,'p,'f,'e)$ *par-confs* = $('s,'p,'f,'e)$ *body* $\times(('s,'p,'f,'e)$ *par-config*$)$
list

inductive-set *par-cptn* :: $('s,'p,'f,'e)$ *par-confs set*
where
ParCptnOne: $(\Gamma,\ [(P,s)]) \in par\text{-}cptn$
| *ParCptnEnv*: $[\![\Gamma\vdash_p(P,s) \rightarrow_e (P,t);(\Gamma,(P,\ t)\#xs) \in par\text{-}cptn\ ]\!] \Longrightarrow(\Gamma,(P,s)\#(P,t)\#xs)$
$\in par\text{-}cptn$
| *ParCptnComp*: $[\![\ \Gamma \vdash_p(P,s) \rightarrow (Q,t);\ (\Gamma,(Q,t)\#xs) \in par\text{-}cptn\ ]\!] \Longrightarrow (\Gamma,(P,s)\#(Q,t)\#xs)$
$\in par\text{-}cptn$

inductive-cases *par-cptn-elim-cases* [*cases set*]:
$(\Gamma,\ [(P,s)]) \in par\text{-}cptn$
$(\Gamma,(P,s)\#(Q,t)\#xs) \in par\text{-}cptn$

lemma *pe-ce*:
assumes *a1*:$\Gamma\vdash_p(P,s) \rightarrow_e (P,t)$
shows $\forall\, i<length\ P.\ \Gamma\vdash_c(P!i,s) \rightarrow_e (P!i,t)$
proof −
{**fix** *i*
assume $i<\ length\ P$
have $\Gamma\vdash_c(P!i,s) \rightarrow_e (P!i,t)$ **using** *a1*
by (*metis Env Env-n env-pe-not-normal-s*)
}
thus $\forall\, i<length\ P.\ \Gamma\vdash_c(P!i,s) \rightarrow_e (P!i,t)$ **by** *blast*

qed

type-synonym $('s,'p,'f,'e)$ *par-com* = $('s,'p,'f,'e)$ *com list*

definition *par-cp* :: $('s,'p,'f,'e)$ *body* ⇒ $('s,'p,'f,'e)$ *com list* ⇒ $('s,'f)$ *xstate* ⇒ $(('s,'p,'f,'e)$ *par-confs*) *set*
where
par-cp Γ P s ≡ $\{(\Gamma 1,l).\ l!0=(P,s) \wedge (\Gamma,l) \in$ *par-cptn* $\wedge\ \Gamma 1=\Gamma\}$

lemma *par-cptn-dest*:$(\Gamma,(P,s)\#(Q,t)\#xs) \in$ *par-cptn* $\Longrightarrow (\Gamma,(Q,t)\#xs)\in$ *par-cptn*
by (*auto dest*: *par-cptn-elim-cases*)

lemmas about single step computation

8.4 Structural Properties of Small Step Computations

lemma *redex-not-Seq*: *redex c* = *Seq c1 c2* $\Longrightarrow P$
apply (*induct c*)
apply *auto*
done
lemma *redex-not-Catch*: *redex c* = *Catch c1 c2* $\Longrightarrow P$
apply (*induct c*)
apply *auto*
done

lemma *no-step-final*:
assumes *step*: $\Gamma\vdash_c(c,s) \rightarrow (c',s')$
shows *final* $(c,s) \Longrightarrow P$
using *step*
by *induct* (*auto simp add*: *final-def*)

lemma *no-step-final'*:
assumes *step*: $\Gamma\vdash_c$ *cfg* $\rightarrow$ *cfg'*
shows *final cfg* $\Longrightarrow P$
using *step*
by (*cases cfg*, *cases cfg'*) (*auto intro*: *no-step-final*)

lemma *step-Abrupt*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
shows $\bigwedge x.\ s=Abrupt\ x \Longrightarrow s'=Abrupt\ x$
using *step*
by (*induct*) *auto*

lemma *step-Fault*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$

  shows $\bigwedge f.\ s{=}\mathit{Fault}\ f \Longrightarrow s'{=}\mathit{Fault}\ f$
using *step*
by (*induct*) *auto*

lemma *step-Stuck*:
  assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
  shows $\bigwedge f.\ s{=}\mathit{Stuck} \Longrightarrow s'{=}\mathit{Stuck}$
using *step*
by (*induct*) *auto*

lemma *step-not-normal-not-normal*:
  assumes *step*:$\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
  shows $\forall\, s1.\ s{\neq}\mathit{Normal}\ s1 \Longrightarrow \forall\, s1.\ s' \neq \mathit{Normal}\ s1$
using *step step-Abrupt step-Stuck step-Fault*
by (*induct*) *auto*

lemma *step-not-normal-s-eq-t*:
  assumes *step*:$\Gamma\vdash_c (c,\ s) \rightarrow (c',\ t)$
  shows $\forall\, s1.\ s{\neq}\mathit{Normal}\ s1 \Longrightarrow s{=}t$
using *step step-Abrupt step-Stuck step-Fault*
by (*induct*) *auto*

lemma *ce-not-normal-s*:
   assumes *a1*:$\Gamma\vdash_c \mathit{cf0} \rightarrow_{ce} \mathit{cf1}$
   shows $\bigwedge\ c_1\ c_2\ s\ s'.\ [\![\mathit{cf0} = (c_1,s);\mathit{cf1}{=}(c_2,s');(\forall\, sa.\ (s{\neq}\mathit{Normal}\ sa))]\!]$
                 $\Longrightarrow s{=}s'$
using *a1*
apply (*clarify*, *cases rule*:*step-ce.cases*)
by (*metis step-not-normal-s-eq-t env-not-normal-s*)+

lemma *SeqSteps*:
  assumes *steps*: $\Gamma\vdash_c \mathit{cfg}_1{\rightarrow}^*\ \mathit{cfg}_2$
  shows $\bigwedge\ c_1\ s\ c_1{'}\ s'.\ [\![\mathit{cfg}_1 = (c_1,s);\mathit{cfg}_2{=}(c_1{'},s')]\!]$
         $\Longrightarrow \Gamma\vdash_c(\mathit{Seq}\ c_1\ c_2,s) \rightarrow^* (\mathit{Seq}\ c_1{'}\ c_2,\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
  case *Refl*
  thus *?case*
    by *simp*
next
  case (*Trans* $\mathit{cfg}_1\ \mathit{cfg}''$)
  have *step*: $\Gamma\vdash_c \mathit{cfg}_1 \rightarrow \mathit{cfg}''$ **using** *Trans.hyps*(*1*) **by** *blast*
  have *steps*: $\Gamma\vdash_c \mathit{cfg}'' \rightarrow^* \mathit{cfg}_2$ **by** *fact*
  have cfg_1: $\mathit{cfg}_1 = (c_1,\ s)$ **and** cfg_2: $\mathit{cfg}_2 = (c_1{'},\ s')$ **by** *fact*+
  obtain $c_1{''}\ s''$ **where** cfg'': $\mathit{cfg}''{=}(c_1{''},s'')$
    by (*cases* cfg'') *auto*
  from *step* $\mathit{cfg}_1\ \mathit{cfg}''$
  have $\Gamma\vdash_c (c_1,s) \rightarrow (c_1{''},s'')$
    by *simp*

hence $\Gamma\vdash_c$ (*Seq* c_1 c_2,*s*) $\rightarrow$ (*Seq* c_1'' c_2,s'') **by** (*simp add*: *Seqc*)
also from *Trans.hyps* (*3*) [*OF* cfg'' cfg_2]
have $\Gamma\vdash_c$ (*Seq* c_1'' c_2, s'') $\rightarrow^*$ (*Seq* c_1' c_2, s') **.**
finally show *?case* **.**
qed

lemma *CatchSteps*:
assumes *steps*: $\Gamma\vdash_c cfg_1 \rightarrow^* cfg_2$
shows $\bigwedge$ c_1 s c_1' s'. $[\![cfg_1 = (c_1,s);\ cfg_2=(c_1',s')]\!]$
$\Longrightarrow \Gamma\vdash_c$(*Catch* c_1 c_2,*s*) $\rightarrow^*$ (*Catch* c_1' c_2, s')
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
case *Refl*
thus *?case*
by *simp*
next
case (*Trans* cfg_1 cfg'')
have *step*: $\Gamma\vdash_c cfg_1 \rightarrow cfg''$ **by** *fact*
have *steps*: $\Gamma\vdash_c cfg'' \rightarrow^* cfg_2$ **by** *fact*
have cfg_1: $cfg_1 = (c_1,\ s)$ **and** cfg_2: $cfg_2 = (c_1',\ s')$ **by** *fact+*
obtain c_1'' s'' **where** cfg'': $cfg''=(c_1'',s'')$
by (*cases* cfg'') *auto*
from *step* cfg_1 cfg''
have *s*: $\Gamma\vdash_c (c_1,s) \rightarrow (c_1'',s'')$
by *simp*
hence $\Gamma\vdash_c$ (*Catch* c_1 c_2,*s*) $\rightarrow$ (*Catch* c_1'' c_2,s'')
by (*rule stepc.Catchc*)
also from *Trans.hyps* (*3*) [*OF* cfg'' cfg_2]
have $\Gamma\vdash_c$ (*Catch* c_1'' c_2, s'') $\rightarrow^*$ (*Catch* c_1' c_2, s') **.**
finally show *?case* **.**
qed

lemma *steps-Fault*: $\Gamma\vdash_c$ (*c*, *Fault f*) $\rightarrow^*$ (*Skip*, *Fault f*)
proof (*induct c*)
case (*Seq* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash_c$ (c_1, *Fault f*) $\rightarrow^*$ (*Skip*, *Fault f*) **by** *fact*
have *steps-*c_2: $\Gamma\vdash_c$ (c_2, *Fault f*) $\rightarrow^*$ (*Skip*, *Fault f*) **by** *fact*
from *SeqSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash_c$ (*Seq* c_1 c_2, *Fault f*) $\rightarrow^*$ (*Seq Skip* c_2, *Fault f*)**.**
also
have $\Gamma\vdash_c$ (*Seq Skip* c_2, *Fault f*) $\rightarrow$ (c_2, *Fault f*) **by** (*rule SeqSkipc*)
also note *steps-*c_2
finally show *?case* **by** *simp*
next
case (*Catch* c_1 c_2)
have *steps-*c_1: $\Gamma\vdash_c$ (c_1, *Fault f*) $\rightarrow^*$ (*Skip*, *Fault f*) **by** *fact*
from *CatchSteps* [*OF steps-*c_1 *refl refl*]
have $\Gamma\vdash_c$ (*Catch* c_1 c_2, *Fault f*) $\rightarrow^*$ (*Catch Skip* c_2, *Fault f*)**.**
also

have $\Gamma\vdash_c$ (*Catch Skip* c_2, *Fault f*) $\rightarrow$ (*Skip*, *Fault f*) **by** (*rule CatchSkipc*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *stepc.intros*)+

lemma *steps-Stuck*: $\Gamma\vdash_c$ (*c*, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*)
proof (*induct c*)
case (*Seq* c_1 c_2)
have *steps-c*$_1$: $\Gamma\vdash_c$ (c_1, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
have *steps-c*$_2$: $\Gamma\vdash_c$ (c_2, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
from *SeqSteps* [*OF steps-c*$_1$ *refl refl*]
have $\Gamma\vdash_c$ (*Seq* c_1 c_2, *Stuck*) $\rightarrow^*$ (*Seq Skip* c_2, *Stuck*)**.**
also
have $\Gamma\vdash_c$ (*Seq Skip* c_2, *Stuck*) $\rightarrow$ (c_2, *Stuck*) **by** (*rule SeqSkipc*)
also note *steps-c*$_2$
finally show *?case* **by** *simp*
next
case (*Catch* c_1 c_2)
have *steps-c*$_1$: $\Gamma\vdash_c$ (c_1, *Stuck*) $\rightarrow^*$ (*Skip*, *Stuck*) **by** *fact*
from *CatchSteps* [*OF steps-c*$_1$ *refl refl*]
have $\Gamma\vdash_c$ (*Catch* c_1 c_2, *Stuck*) $\rightarrow^*$ (*Catch Skip* c_2, *Stuck*) **.**
also
have $\Gamma\vdash_c$ (*Catch Skip* c_2, *Stuck*) $\rightarrow$ (*Skip*, *Stuck*) **by** (*rule CatchSkipc*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *stepc.intros*)+

lemma *steps-Abrupt*: $\Gamma\vdash_c$ (*c*, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*)
proof (*induct c*)
case (*Seq* c_1 c_2)
have *steps-c*$_1$: $\Gamma\vdash_c$ (c_1, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
have *steps-c*$_2$: $\Gamma\vdash_c$ (c_2, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
from *SeqSteps* [*OF steps-c*$_1$ *refl refl*]
have $\Gamma\vdash_c$ (*Seq* c_1 c_2, *Abrupt s*) $\rightarrow^*$ (*Seq Skip* c_2, *Abrupt s*)**.**
also
have $\Gamma\vdash_c$ (*Seq Skip* c_2, *Abrupt s*) $\rightarrow$ (c_2, *Abrupt s*) **by** (*rule SeqSkipc*)
also note *steps-c*$_2$
finally show *?case* **by** *simp*
next
case (*Catch* c_1 c_2)
have *steps-c*$_1$: $\Gamma\vdash_c$ (c_1, *Abrupt s*) $\rightarrow^*$ (*Skip*, *Abrupt s*) **by** *fact*
from *CatchSteps* [*OF steps-c*$_1$ *refl refl*]
have $\Gamma\vdash_c$ (*Catch* c_1 c_2, *Abrupt s*) $\rightarrow^*$ (*Catch Skip* c_2, *Abrupt s*)**.**
also
have $\Gamma\vdash_c$ (*Catch Skip* c_2, *Abrupt s*) $\rightarrow$ (*Skip*, *Abrupt s*) **by** (*rule CatchSkipc*)
finally show *?case* **by** *simp*
qed (*fastforce intro*: *stepc.intros*)+

lemma *step-Fault-prop*:
assumes *step*: $\Gamma\vdash_c$ (*c*, *s*) $\rightarrow$ (c', s')

shows $\bigwedge f.\ s{=}Fault\ f \Longrightarrow s'{=}Fault\ f$
using *step*
by (*induct*) *auto*

lemma *step-Abrupt-prop*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
shows $\bigwedge x.\ s{=}Abrupt\ x \Longrightarrow s'{=}Abrupt\ x$
using *step*
by (*induct*) *auto*

lemma *step-Stuck-prop*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
shows $s{=}Stuck \Longrightarrow s'{=}Stuck$
using *step*
by (*induct*) *auto*

lemma *steps-Fault-prop*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s')$
shows $s{=}Fault\ f \Longrightarrow s'{=}Fault\ f$
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *simp*
next
case (*Trans c s c'' s''*)
thus *?case* **by** (*simp add*: *step-Fault-prop*)
qed

lemma *steps-Abrupt-prop*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s')$
shows $s{=}Abrupt\ t \Longrightarrow s'{=}Abrupt\ t$
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *simp*
next
case (*Trans c s c'' s''*)
thus *?case*
by (*auto intro*: *step-Abrupt-prop*)
qed

lemma *steps-Stuck-prop*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s')$
shows $s{=}Stuck \Longrightarrow s'{=}Stuck$
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *simp*
next
case (*Trans c s c'' s''*)
thus *?case*
by (*auto intro*: *step-Stuck-prop*)

qed

lemma *step-seq-throw-normal*:
assumes *step*: $\Gamma \vdash_c (c,\ s) \rightarrow (c',\ s')$ **and**
c-val: $c{=}Seq\ Throw\ Q \wedge c'{=}Throw$
shows $\exists sa.\ s{=}Normal\ sa$
using *step c-val*
proof (*cases s*)
case *Normal*
thus $\exists sa.\ s{=}Normal\ sa$ **by** *auto*
next
case *Abrupt*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*5*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
next
case *Stuck*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*5*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
next
case *Fault*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*5*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
qed

lemma *step-catch-throw-normal*:
assumes *step*: $\Gamma \vdash_c (c,\ s) \rightarrow (c',\ s')$ **and**
c-val: $c{=}Catch\ Throw\ Q \wedge c'{=}Throw$
shows $\exists sa.\ s{=}Normal\ sa$
using *step c-val*
proof (*cases s*)
case *Normal*
thus $\exists sa.\ s{=}Normal\ sa$ **by** *auto*
next
case *Abrupt*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*12*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
next
case *Stuck*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*12*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
next
case *Fault*
thus $\exists sa.\ s{=}Normal\ sa$ **using** *step c-val stepc-elim-cases*(*12*)[*of* Γ *Throw Q s* (*Throw*,*s'*)] **by** *auto*
qed

lemma *step-normal-to-normal*[*rule-format*]:
assumes *step*:$\Gamma \vdash_c (c,\ s) \rightarrow^* (c',\ s')$ **and**

```
       sn: s = Normal sa and
       finalc':(Γ⊢c (c', s') →*(c1, s1) ∧  (∃ sb. s1 = Normal sb))
shows  (∃ sc. s'=Normal sc)
using step sn finalc'
 proof (induct arbitrary: sa rule: converse-rtranclp-induct2 [case-names Refl Trans])
   case Refl show ?case by (simp add: Refl.prems)
 next
   case (Trans c s c'' s'') thm converse-rtranclpE2
     thus ?case
     proof (cases s'')
       case (Abrupt a1) thus ?thesis using finalc' by (metis steps-Abrupt-prop
Trans.hyps(2))
     next
    case Stuck thus ?thesis using finalc' by (metis steps-Stuck-prop Trans.hyps(2))

     next
    case Fault thus ?thesis using finalc' by (metis steps-Fault-prop Trans.hyps(2))

     next
      case Normal thus ?thesis using Trans.hyps(3) finalc' by blast
    qed
qed

lemma step-spec-skip-normal-normal:
  assumes a0:Γ⊢c (c,s)  → (c',s') and
          a1:c=Spec r e and
          a2: s=Normal s1 and
          a3: c'=Skip and
          a4: (∃ t. (s1,t) ∈ r)
  shows ∃ s1'. s'=Normal s1'
proof (cases s')
  case (Normal u) thus ?thesis by auto
next
  case Stuck
    have ∀ f r b p e. ¬ f⊢c (LanguageCon.com.Spec r e, Normal b) → p ∨
           (∃ ba. p = (Skip::('b, 'a, 'c,'d) com, Normal ba) ∧ (b, ba) ∈ r) ∨
           p = (Skip, Stuck) ∧ (∀ ba. (b, ba) ∉ r)
      by (meson stepc-Normal-elim-cases(4))
      thus ?thesis using a0 a1 a2 a4 by blast
next
  case (Fault f)
  have ∀ f r b p e. ¬ f⊢c (LanguageCon.com.Spec r e, Normal b) → p ∨
           (∃ ba. p = (Skip::('b, 'a, 'c,'d) com, Normal ba) ∧ (b, ba) ∈ r) ∨
           p = (Skip, Stuck) ∧ (∀ ba. (b, ba) ∉ r)
    by (meson stepc-Normal-elim-cases(4))
    thus ?thesis using a0 a1 a2 a4 by blast
next
  have ∀ f r b p e. ¬ f⊢c (LanguageCon.com.Spec r e, Normal b) → p ∨
        (∃ ba. p = (Skip::('b, 'a, 'c,'d) com, Normal ba) ∧ (b, ba) ∈ r) ∨
```

$p = (Skip,\ Stuck) \wedge (\forall\, ba.\ (b,\ ba) \notin r)$
by (*meson stepc-Normal-elim-cases*(*4*))
thus *?thesis* **using** *a0 a1 a2 a4* **by** *blast*
qed

if not Normal not environmental

lemma *no-advance-seq*:
assumes *a0*: $P = Seq\ p1\ p2$ **and**
a1: $\Gamma\vdash_c (p1,Normal\ s) \rightarrow (p1,\ Normal\ s)$
shows $\Gamma\vdash_c (P,Normal\ s) \rightarrow (P,\ Normal\ s)$
by (*simp add*: *Seqc a0 a1*)

lemma *no-advance-catch*:
assumes *a0*: $P = Catch\ p1\ p2$ **and**
a1: $\Gamma\vdash_c (p1,Normal\ s) \rightarrow (p1,\ Normal\ s)$
shows $\Gamma\vdash_c (P,Normal\ s) \rightarrow (P,\ Normal\ s)$
by (*simp add*: *Catchc a0 a1*)

lemma *not-step-c-env*:
$\Gamma\vdash_c (c,\ s) \rightarrow_e (c,\ s') \Longrightarrow$
$(\bigwedge sa.\ \neg(s=Normal\ sa)) \Longrightarrow$
$(\bigwedge sa.\ \neg(s'=Normal\ sa))$
by (*fastforce elim*:*stepe-elim-cases*)

lemma *step-c-env-not-normal-eq-state*:
$\Gamma\vdash_c (c,\ s) \rightarrow_e (c,\ s') \Longrightarrow$
$(\bigwedge sa.\ \neg(s=Normal\ sa)) \Longrightarrow$
$s=s'$
by (*fastforce elim*:*stepe-elim-cases*)

lemma *not-eq-not-env*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_{ce} (c',\ s')$
shows $^\sim(c=c') \Longrightarrow \Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s') \Longrightarrow P$
using *step-m etranE* **by** *blast*

lemma *step-ce-not-step-e-step-c*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_{ce} (c',\ s')$
shows $\neg\ (\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s')) \Longrightarrow (\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s'))$
using *step-m step-ce-elim-cases* **by** *blast*

lemma *step-ce-notNormal*:
assumes *step-m*: $\Gamma\vdash_c (c,\ s) \rightarrow_{ce} (c',\ s')$
shows $(\forall\, sa.\ \neg(s=Normal\ sa)) \Longrightarrow s'=s$
using *step-m*
proof (*induct rule*:*step-ce-induct*)
case (*e-step a b a' b'*)
have $\forall\, f\, p\, pa.\ \neg\, f\vdash_c p \rightarrow_e pa \vee (\exists\, c.\ (\exists\, x.\ p = (c::('b,\ 'a,\ 'c,'d)\ LanguageCon.com,\ x)) \wedge (\exists\, x.\ pa = (c,\ x)))$

by (*fastforce elim*:*etranE stepe-elim-cases*)
thus *?case*
using *stepe-elim-cases e-step.hyps e-step.prems* **by** *blast*
next
case (*c-step a b a′ b′*)
thus *?case*
proof (*cases b*)
case (*Normal*) **thus** *?thesis* **using** *c-step.prems* **by** *auto*
next
case (*Stuck*) **thus** *?thesis*
using *SmallStepCon.step-Stuck-prop c-step.hyps* **by** *blast*
next
case (*Fault f*) **thus** *?thesis*
using *SmallStepCon.step-Fault-prop c-step.hyps* **by** *fastforce*
next
case (*Abrupt a*) **thus** *?thesis*
using *SmallStepCon.step-Abrupt-prop c-step.hyps* **by** *fastforce*
qed
qed

lemma *steps-ce-not-Normal*:
assumes *step-m*: $\Gamma\vdash_c (c, s) \rightarrow_{ce}{}^* (c', s')$
shows $\forall sa.\ \neg(s=Normal\ sa) \Longrightarrow s'=s$
using *step-m*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **then show** *?case* **by** *auto*
next
case (*Trans a b a′ b′*)
thus *?case* **using** *step-ce-notNormal* **by** *blast*
qed

lemma *steps-not-normal-ce-c*:
assumes *steps*: $\Gamma\vdash_c (c, s) \rightarrow_{ce}{}^* (c', s')$
shows $(\forall sa.\ \neg(s=Normal\ sa)) \Longrightarrow \Gamma\vdash_c (c, s) \rightarrow^* (c', s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *auto*
next
case (*Trans a b a′ b′*)
then have $b=b'$ **using** *step-ce-notNormal* **by** *blast*
then have $\Gamma\vdash_c (a', b') \rightarrow^* (c', s')$ **using** ‹$b=b'$› *Trans.hyps*(*3*) *Trans.prems*
by *blast*
then have $\Gamma\vdash_c (a, b) \rightarrow (a', b') \vee \Gamma\vdash_c (a, b) \rightarrow_e (a', b')$
using *Trans.hyps*(*1*) **by** (*fastforce elim*: *step-ce-elim-cases*)
thus *?case*
proof
assume $\Gamma\vdash_c (a, b) \rightarrow (a', b')$
thus *?thesis* **using** ‹$\Gamma\vdash_c (a', b') \rightarrow^* (c', s')$› **by** *auto*
next

assume $\Gamma\vdash_c (a,\ b) \rightarrow_e (a',\ b')$
have $a = a'$
by (*meson Trans.hyps*(*1*) ‹$\Gamma\vdash_c (a,\ b) \rightarrow_e (a',\ b')$› *not-eq-not-env*)
thus *?thesis* **using** ‹$\Gamma\vdash_c (a',\ b') \rightarrow^* (c',\ s')$› ‹$b = b'$› **by** *force*
qed
qed

lemma *steps-c-ce*:
assumes *steps*: $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s')$
shows $\Gamma\vdash_c (c,\ s) \rightarrow_{ce}{}^* (c',\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl* **thus** *?case* **by** *auto*
next
case (*Trans a b a′ b′*)
have $\Gamma\vdash_c (a,\ b) \rightarrow_{ce} (a',\ b')$
using *Trans.hyps*(*1*) *c-step* **by** *blast*
thus *?case*
by (*simp add*: *Trans.hyps*(*3*) *converse-rtranclp-into-rtranclp*)
qed

lemma *steps-not-normal-c-ce*:
assumes *steps*: $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s')$
shows $(\forall sa.\ \neg(s{=}Normal\ sa)) \Longrightarrow \Gamma\vdash_c (c,\ s) \rightarrow_{ce}{}^* (c',\ s')$
by (*simp add*: *steps steps-c-ce*)

lemma *steps-not-normal-c-eq-ce*:
assumes *normal*: $(\forall sa.\ \neg(s{=}Normal\ sa))$
shows $\Gamma\vdash_c (c,\ s) \rightarrow^* (c',\ s') = \Gamma\vdash_c (c,\ s) \rightarrow_{ce}{}^* (c',\ s')$
using *normal*
using *steps-c-ce steps-not-normal-ce-c* **by** *auto*

lemma *steps-ce-Fault*: $\Gamma\vdash_c (c,\ Fault\ f) \rightarrow_{ce}{}^* (Skip,\ Fault\ f)$
by (*simp add*: *SmallStepCon.steps-Fault steps-c-ce*)

lemma *steps-ce-Stuck*: $\Gamma\vdash_c (c,\ Stuck) \rightarrow_{ce}{}^* (Skip,\ Stuck)$
by (*simp add*: *SmallStepCon.steps-Stuck steps-c-ce*)

lemma *steps-ce-Abrupt*: $\Gamma\vdash_c (c,\ Abrupt\ a) \rightarrow_{ce}{}^* (Skip,\ Abrupt\ a)$
by (*simp add*: *SmallStepCon.steps-Abrupt steps-c-ce*)

lemma *step-ce-seq-throw-normal*:
assumes *step*: $\Gamma\vdash_c (c,\ s) \rightarrow_{ce} (c',\ s')$ **and**
c-val: $c{=}Seq\ Throw\ Q \wedge c'{=}Throw$
shows $\exists sa.\ s{=}Normal\ sa$
using *step c-val not-eq-not-env*
step-ce-not-step-e-step-c step-seq-throw-normal **by** *blast*

lemma *step-ce-catch-throw-normal*:

assumes *step*: $\Gamma\vdash_c (c, s) \rightarrow_{ce} (c', s')$ **and**
c-val: c=*Catch Throw Q* $\wedge$ c'=*Throw*
shows $\exists$ *sa*. *s*=*Normal sa*
using *step c-val not-eq-not-env*
step-ce-not-step-e-step-c step-catch-throw-normal **by** *blast*

lemma *step-ce-normal-to-normal*[*rule-format*]:
assumes *step*:$\Gamma\vdash_c (c, s) \rightarrow_{ce}{}^{*} (c', s')$ **and**
sn: *s* = *Normal sa* **and**
finalc':($\Gamma\vdash_c (c', s') \rightarrow_{ce}{}^{*}(c1, s1)$ $\wedge$ ($\exists$ *sb*. *s1* = *Normal sb*))
shows
($\exists$ *sc*. *s'*=*Normal sc*)
using *step sn finalc' steps-ce-not-Normal* **by** *blast*

lemma *SeqSteps-ce*:
assumes *steps*: $\Gamma\vdash_c cfg_1 \rightarrow_{ce}{}^{*} cfg_2$
shows $\bigwedge c_1\ s\ c_1'\ s'.$ $[\![cfg_1 = (c_1,s);cfg_2=(c_1',s')]\!]$
$\Longrightarrow \Gamma\vdash_c(Seq\ c_1\ c_2,s) \rightarrow_{ce}{}^{*} (Seq\ c_1'\ c_2,\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
case *Refl*
thus *?case*
by *simp*
next
case (*Trans* cfg_1 cfg'')
then have $\Gamma\vdash_c cfg_1 \rightarrow cfg'' \vee \Gamma\vdash_c cfg_1 \rightarrow_e cfg''$
using *step-ce-elim-cases* **by** *blast*
thus *?case*
proof
assume *a1*:$\Gamma\vdash_c cfg_1 \rightarrow_e cfg''$
have $\forall f\ p\ pa.\ \neg\ f\vdash_c p \rightarrow_e pa \vee (\exists c.$
$(\exists x.\ p = (c::('a, 'b, 'c,'d)\ LanguageCon.com,\ x)) \wedge (\exists x.\ pa = (c,$
$x)))$
by (*meson etranE*)
then obtain *cc* :: ('b $\Rightarrow$ ('a, 'b, 'c,'d) *LanguageCon.com option*) $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* **and**
xx :: ('b $\Rightarrow$ ('a, 'b, 'c,'d) *LanguageCon.com option*) $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$ ('a, 'c)
xstate **and**
xxa :: ('b $\Rightarrow$ ('a, 'b, 'c,'d) *LanguageCon.com option*) $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$
('a, 'b, 'c,'d) *LanguageCon.com* $\times$ ('a, 'c) *xstate* $\Rightarrow$ ('a, 'c)
xstate **where**
f1: $\forall f\ p\ pa.\ \neg\ f\vdash_c p \rightarrow_e pa \vee p = (cc\ f\ p\ pa,\ xx\ f\ p\ pa) \wedge pa = (cc\ f\ p\ pa,$
$xxa\ f\ p\ pa)$
by (*metis* (*no-types*))

have $f2$: $\forall f\ c\ x\ xa.\ \neg\ f\vdash_c\ (c{::}('a,\ 'b,\ 'c,'d)\ LanguageCon.com,\ x) \rightarrow_e\ (c,\ xa)$
$\vee$
$(\exists\, a.\ x = Normal\ a) \vee (\forall\, a.\ xa \neq Normal\ a) \wedge x = xa$
by (*metis stepe-elim-cases*)
have $f3$: $(c_1,\ xxa\ \Gamma\ cfg_1\ cfg'') = cfg''$
using *f1* **by** (*metis Trans.prems(1) a1 fst-conv*)
hence $\Gamma\vdash_c (LanguageCon.com.Seq\ c_1\ c_2,\ xxa\ \Gamma\ cfg_1\ cfg'') \rightarrow_{ce}{}^*\ (LanguageCon.com.Seq\ c_1{}'\ c_2,\ s')$
using *Trans.hyps(3) Trans.prems(2)* **by** *force*
thus *?thesis*
using *f3 f2* **by** (*metis (no-types) Env Trans.prems(1) a1 e-step r-into-rtranclp*

rtranclp.rtrancl-into-rtrancl rtranclp-idemp)
next
assume $\Gamma\vdash_c\ cfg_1 \rightarrow cfg''$
thus *?thesis*
proof −
have $\forall\, p.\ \exists\, c\ x.\ p = (c{::}('a,\ 'b,\ 'c,'d)\ LanguageCon.com,\ x{::}('a,\ 'c)\ xstate)$
by *auto*
thus *?thesis*
by (*metis (no-types) Seqc Trans.hyps(3) Trans.prems(1) Trans.prems(2)*
$\langle\Gamma\vdash_c\ cfg_1 \rightarrow cfg''\rangle$ *c-step converse-rtranclp-into-rtranclp*)
qed
qed
qed

lemma *CatchSteps-ce*:
assumes *steps*: $\Gamma\vdash_c cfg_1 \rightarrow_{ce}{}^*\ cfg_2$
shows $\bigwedge c_1\ s\ c_1{}'\ s'.\ [\![cfg_1 = (c_1,s);\ cfg_2 = (c_1{}',s')]\!]$
$\Longrightarrow \Gamma\vdash_c(Catch\ c_1\ c_2,s) \rightarrow_{ce}{}^*\ (Catch\ c_1{}'\ c_2,\ s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct* [*case-names Refl Trans*])
case *Refl*
thus *?case*
by *simp*
next
case (*Trans* $cfg_1\ cfg''$)
then have $\Gamma\vdash_c\ cfg_1 \rightarrow cfg'' \vee \Gamma\vdash_c\ cfg_1 \rightarrow_e\ cfg''$
using *step-ce-elim-cases* **by** *blast*
thus *?case*
proof
assume $a1$:$\Gamma\vdash_c\ cfg_1 \rightarrow_e\ cfg''$
have $\forall f\ p\ pa.\ \neg\ f\vdash_c\ p \rightarrow_e\ pa \vee (\exists\, c.\ (\exists\, x.\ p = (c{::}('a,\ 'b,\ 'c,'d)\ LanguageCon.com,\ x)) \wedge (\exists\, x.\ pa = (c,\ x)))$
by (*meson etranE*)
then obtain $cc :: ('b \Rightarrow ('a,\ 'b,\ 'c,\ 'd)\ LanguageCon.com\ option) \Rightarrow$
$('a,\ 'b,\ 'c,\ 'd)\ LanguageCon.com \times ('a,\ 'c)\ xstate \Rightarrow$
$('a,\ 'b,\ 'c,\ 'd)\ LanguageCon.com \times ('a,\ 'c)\ xstate \Rightarrow$

('a, 'b, 'c, 'd) *LanguageCon.com* **and**
xx :: ('b ⇒('a, 'b, 'c, 'd) *LanguageCon.com option*) ⇒
('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒
('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒
('a, 'c) *xstate* **and**
xxa :: ('b ⇒('a, 'b, 'c, 'd) *LanguageCon.com option*) ⇒
('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒
('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒ ('a, 'c)
xstate **where**
f1: ∀ *f p pa*. ¬ *f*$\vdash_c$ *p* $\rightarrow_e$ *pa* ∨ *p* = (*cc f p pa*, *xx f p pa*) ∧ *pa* = (*cc f p pa*, *xxa f p pa*)
by (*metis* (*no-types*))
have *f2*: ∀ *f c x xa*. ¬ *f*$\vdash_c$ (*c*::('a, 'b, 'c,'d) *LanguageCon.com*, *x*) $\rightarrow_e$ (*c*, *xa*) ∨
(∃ *a*. *x* = *Normal a*) ∨ (∀ *a*. *xa* ≠ *Normal a*) ∧ *x* = *xa*
by (*metis stepe-elim-cases*)
have *f3*: (c_1, *xxa* Γ cfg_1 *cfg''*) = *cfg''*
using *f1* **by** (*metis Trans.prems*(*1*) *a1 fst-conv*)
hence Γ$\vdash_c$ (*LanguageCon.com.Catch* c_1 c_2, *xxa* Γ cfg_1 *cfg''*) $\rightarrow_{ce}{}^*$ (*LanguageCon.com.Catch* $c_1{}'$ c_2, *s'*)
using *Trans.hyps*(*3*) *Trans.prems*(*2*) **by** *force*
thus *?thesis*
using *f3 f2* **by** (*metis* (*no-types*) *Env Trans.prems*(*1*) *a1 e-step r-into-rtranclp rtranclp.rtrancl-into-rtrancl rtranclp-idemp*)
next
assume Γ$\vdash_c$ cfg_1 → *cfg''*
thus *?thesis*
proof −
obtain *cc* :: ('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒ ('a, 'b, 'c, 'd) *LanguageCon.com* **and** *xx* :: ('a, 'b, 'c, 'd) *LanguageCon.com* × ('a, 'c) *xstate* ⇒ ('a, 'c) *xstate* **where**
f1: ∀ *p*. *p* = (*cc p*, *xx p*)
by (*meson old.prod.exhaust*)
hence ⋀*c*. Γ$\vdash_c$ (*LanguageCon.com.Catch* c_1 *c*, *s*) → (*LanguageCon.com.Catch* (*cc cfg''*) *c*, *xx cfg''*)
by (*metis* (*no-types*) *Catchc Trans.prems*(*1*) ‹Γ$\vdash_c$ cfg_1 → *cfg''*›)
thus *?thesis*
using *f1* **by** (*meson Trans.hyps*(*3*) *Trans.prems*(*2*) *c-step converse-rtranclp-into-rtranclp*)
qed
qed
qed

lemma *step-change-p-or-eq-Ns*:
assumes *step*: Γ$\vdash_c$ (*P*,*Normal s*) → (*Q*,*s'*)
shows ¬(*P*=*Q*)
using *step*
proof (*induct P arbitrary*: *Q s s'*)
qed(*fastforce elim*: *stepc-Normal-elim-cases*)+

lemma *step-change-p-or-eq-s*:
  assumes *step*: $\Gamma\vdash_c (P,s) \rightarrow (Q,s')$
  shows $\neg(P=Q)$
using *step*
proof (*induct P arbitrary*: *Q s s'*)
qed (*fastforce elim*: *stepc-elim-cases*)+

8.5 Relation between *stepc-rtrancl* and *cptn*

lemma *stepc-rtrancl-cptn*:
 assumes *step*: $\Gamma\vdash_c (c,s) \rightarrow_{ce}{}^* (cf,sf)$
 shows $\exists xs.\ (\Gamma,(c,\ s)\#xs) \in cptn \wedge (cf,sf) = (last\ ((c,s)\#xs))$
using *step*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
 case *Refl* **thus** *?case* **using** *cptn.CptnOne* **by** *auto*
next
 case (*Trans c s c' s'*)
 have $\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s') \vee \Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
  by (*meson Trans.hyps*(*1*) *step-ce.simps*)
 then show *?case*
 proof
  assume *prem*:$\Gamma\vdash_c (c,\ s) \rightarrow_e (c',\ s')$
  then have *ceqc'*:$c=c'$ **using** *prem env-c-c'*
   by *auto*
  obtain *xs* **where** *xs-s*:$(\Gamma,\ (c',\ s')\ \#\ xs) \in cptn \wedge (cf,\ sf) = last\ ((c',\ s')\ \#$
$xs)$
   using *Trans*(*3*) **by** *auto*
  then have *xs-f*: $(\Gamma,\ (c,\ s)\#(c',\ s')\ \#\ xs) \in cptn$
  using *cptn.CptnEnv ceqc' prem* **by** *fastforce*
  also have $last\ ((c',\ s')\ \#\ xs) = last\ ((c,s)\#(c',\ s')\ \#\ xs)$ **by** *auto*
  then have $(cf,\ sf) = last\ ((c,\ s)\ \#\ (c',\ s')\ \#\ xs)$
   using *xs-s* **by** *auto*
  thus *?thesis*
   using *xs-f* **by** *blast*
 next
  assume *prem*:$\Gamma\vdash_c (c,\ s) \rightarrow (c',\ s')$
  obtain *xs* **where** *xs-s*:$(\Gamma,\ (c',\ s')\ \#\ xs) \in cptn \wedge (cf,\ sf) = last\ ((c',\ s')\ \#$
$xs)$
   using *Trans*(*3*) **by** *auto*
  have $(\Gamma,\ (c,\ s)\ \#\ (c',\ s')\ \#\ xs) \in cptn$ **using** *cptn.CptnComp*
   using *xs-s prem* **by** *blast*
  also have $last\ ((c',\ s')\ \#\ xs) = last\ ((c,s)\#(c',\ s')\ \#\ xs)$ **by** *auto*
  ultimately show *?thesis* **using** *xs-s* **by** *fastforce*
 qed
qed

lemma *cptn-stepc-rtrancl*:

assumes *cptn-step*: $(\Gamma,(c,\ s)\#xs) \in cptn$ **and**
cf-last:$(cf,sf) = (last\ ((c,s)\#xs))$
shows $\Gamma\vdash_c (c,s) \rightarrow_{ce}{}^* (cf,sf)$
using *cptn-step cf-last*
proof (*induct xs arbitrary*: *c s*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons a xs c s*)
then obtain *ca sa* **where** *eq-pair*: $a=(ca,sa)$ **and** $(cf,\ sf) = last\ ((ca,sa)\ \#\ xs)$

using *Cons* **by** (*fastforce*)
have *f1*: $\forall f\ p\ pa.\ \neg\ (f::'a \Rightarrow ('b,\ \text{-},\ 'c,'d)\ LanguageCon.com\ option)\vdash_c p \rightarrow pa \vee f\vdash_c p \rightarrow_{ce} pa$
by (*simp add*: *c-step*)
have *f2*: $(\Gamma,\ (c,\ s)\ \#\ (ca,\ sa)\ \#\ xs) \in cptn$
using ‹$(\Gamma,\ (c,\ s)\ \#\ a\ \#\ xs) \in cptn$› *eq-pair* **by** *blast*
have *f3*: $\forall f\ p\ pa.\ \neg\ (f::'a \Rightarrow ('b,\ \text{-},\ 'c,'d)\ LanguageCon.com\ option)\vdash_c p \rightarrow_e pa \vee f\vdash_c p \rightarrow_{ce} pa$
using *e-step* **by** *blast*
have $\forall c\ x.\ (\Gamma,\ (c,\ x)\ \#\ xs) \notin cptn \vee (cf,\ sf) \neq last\ ((c,\ x)\ \#\ xs) \vee \Gamma\vdash_c (c,\ x) \rightarrow_{ce}{}^* (cf,\ sf)$
using *Cons.hyps* **by** *blast*
thus *?case*
using *f3 f2 f1* **by** (*metis* (*no-types*) ‹$(cf,\ sf) = last\ ((ca,\ sa)\ \#\ xs)$› *converse-rtranclp-into-rtranclp cptn-elim-cases*(*2*))
qed

lemma *three-elems-list*:
assumes *a1*:$length\ l > 2$
shows $\exists\ a0\ a1\ a2\ l1.\ l{=}a0\#a1\#a2\#l1$
using *a1* **by** (*metis Cons-nth-drop-Suc One-nat-def Suc-1 Suc-leI add-lessD1 drop-0 length-greater-0-conv list.size*(*3*) *not-numeral-le-zero one-add-one*)

lemma *cptn-stepc-rtran*:
assumes *cptn-step*: $(\Gamma,x\#xs) \in cptn$ **and**
a1:$Suc\ i < length\ (x\#xs)$
shows $\Gamma\vdash_c ((x\#xs)!i) \rightarrow_{ce} ((x\#xs)!(Suc\ i))$
using *cptn-step a1*
proof (*induct i arbitrary*: *x xs*)
case *0*
then obtain *x1 xs1* **where** *xs*:$xs{=}x1\#xs1$
by (*metis length-Cons less-not-refl list.exhaust list.size*(*3*))
then have $(x\#x1\#xs1)!Suc\ 0 = x1$ **by** *fastforce*
have *x-x1-cptn*:$(\Gamma,x\#x1\#xs1){\in}cptn$ **using** *0 xs* **by** *auto*
then have $(\Gamma,x1\#xs1){\in}cptn$
using *cptn-dest-pair* **by** *fastforce*
then have $\Gamma\vdash_c x \rightarrow_e x1 \vee \Gamma\vdash_c x \rightarrow x1$
using *cptn-elim-cases-pair x-x1-cptn* **by** *blast*

```
    then have Γ⊢c x →ce x1
      by (metis c-step e-step)
    then show ?case
      by (simp add: xs)
next
   case (Suc i)
    then have Suc i < length xs by auto
    moreover then obtain x1 xs1 where xs:xs=x1#xs1
      by (metis (full-types) list.exhaust list.size(3) not-less0)
    moreover then have (Γ,x1#xs1) ∈ cptn using Suc cptn-dest-pair by blast
    ultimately have Γ⊢c ((x1 # xs1) ! i) →ce ((x1 # xs1) ! Suc i)
      using Suc by auto
    thus ?case using Suc xs by auto
qed


lemma cptn-stepconf-rtrancl:
  assumes cptn-step: (Γ,cfg1#xs) ∈ cptn and
          cf-last:cfg2 = (last (cfg1#xs))
  shows Γ⊢c cfg1 →ce* cfg2
using cptn-step cf-last
by (metis cptn-stepc-rtrancl prod.collapse)

lemma cptn-all-steps-rtrancl:
  assumes cptn-step: (Γ,cfg1#xs) ∈ cptn
  shows ∀ i<length (cfg1#xs). Γ⊢c cfg1 →ce* ((cfg1#xs)!i)
using cptn-step
proof (induct xs arbitrary: cfg1)
  case Nil thus ?case by auto
next
  case (Cons x xs1) thus ?case
  proof −
     have hyp:∀ i<length (x # xs1). Γ⊢c x →ce* ((x # xs1) ! i)
       using Cons.hyps Cons.prems cptn-dest-pair by blast
     thus ?thesis
     proof
     {
        fix i
        assume a0:i<length (cfg1 # x # xs1)
        then have Suc 0 < length (cfg1 # x # xs1)
          by simp
        hence Γ⊢c (cfg1 # x # xs1) ! 0 →ce ((cfg1 # x # xs1) ! Suc 0)
          using Cons.prems cptn-stepc-rtran by blast
        then have Γ⊢c cfg1 →ce x using Cons by simp
        also have  i < Suc (Suc (length xs1))
          using a0 by force
        ultimately have Γ⊢c cfg1 →ce* (cfg1 # x # xs1) ! i using hyp Cons
         using converse-rtranclp-into-rtranclp hyp less-Suc-eq-0-disj
         by auto
```

```
    } thus ?thesis by auto qed
  qed
qed

lemma cptn-env-same-prog:
assumes a0: (Γ, l) ∈ cptn and
        a1: ∀ k < j. (Γ⊢c(l!k)  →e (l!(Suc k))) and
        a2: Suc j < length l
shows fst (l!j) = fst (l!0)
using a0 a1 a2
proof (induct j arbitrary: l)
  case 0 thus ?case by auto
next
  case (Suc j)
    then have fst (l!j) = fst (l!0) by fastforce
    thus ?case using Suc
      by (metis (no-types) env-c-c' lessI prod.collapse)
qed

lemma takecptn-is-cptn [rule-format, elim!]:
  ∀ j. (Γ,c) ∈ cptn ⟶ (Γ, take (Suc j) c) ∈ cptn
apply(induct c)
 apply(force elim: cptn.cases)
apply clarify
apply(case-tac j)
 apply simp
 apply(rule CptnOne)
apply simp
apply(force intro:cptn.intros elim:cptn.cases)
done



lemma dropcptn-is-cptn [rule-format,elim!]:
  ∀ j<length c. (Γ,c) ∈ cptn ⟶ (Γ, drop j c) ∈ cptn
apply(induct c)
 apply(force elim: cptn.cases)
apply clarify
apply(case-tac j,simp+)
apply(erule cptn.cases)
  apply simp
 apply force
apply force
done

lemma takepar-cptn-is-par-cptn [rule-format,elim]:
  ∀ j. (Γ,c) ∈ par-cptn ⟶ (Γ,take (Suc j) c) ∈ par-cptn
apply(induct c)
 apply(force elim: cptn.cases)
```

apply *clarify*
apply(*case-tac j*,*simp*)
 apply(*rule ParCptnOne*)
apply(*force intro*:*par-cptn.intros elim*:*par-cptn.cases*)
done

lemma *droppar-cptn-is-par-cptn* [*rule-format*]:
  $\forall j<$*length c*. $(\Gamma,c) \in$ *par-cptn* $\longrightarrow$ $(\Gamma,$*drop j c*$) \in$ *par-cptn*
apply(*induct c*)
 apply(*force elim*: *par-cptn.cases*)
apply *clarify*
apply(*case-tac j*,*simp+*)
apply(*erule par-cptn.cases*)
  apply *simp*
 apply *force*
apply *force*
done

8.6 Modular Definition of Computation

definition *lift* :: $('s,'p,'f,'e)$ *com* $\Rightarrow$ $('s,'p,'f,'e)$ *config* $\Rightarrow$ $('s,'p,'f,'e)$ *config* **where**
  lift Q $\equiv \lambda(P,\ s).\ \ ((Seq\ P\ Q),\ s)$

definition *lift-catch* :: $('s,'p,'f,'e)$ *com* $\Rightarrow$ $('s,'p,'f,'e)$ *config* $\Rightarrow$ $('s,'p,'f,'e)$ *config*
where
  lift-catch Q $\equiv \lambda(P,\ s).\ \ (Catch\ P\ Q,\ s)$

inductive-set *cptn-mod* :: $(('s,'p,'f,'e)$ *confs*$)$ *set*
where
  CptnModOne: $(\Gamma,[(P,\ s)]) \in$ *cptn-mod*
| *CptnModEnv*: $[\![\Gamma\vdash_c(P,s) \rightarrow_e (P,t);(\Gamma,(P,\ t)\#xs) \in$ *cptn-mod* $]\!] \Longrightarrow$
  $(\Gamma,(P,\ s)\#(P,\ t)\#xs) \in$ *cptn-mod*
| *CptnModSkip*: $[\![\Gamma\vdash_c(P,s) \rightarrow (Skip,t);$ *redex* $P = P;$
  $(\Gamma,(Skip,\ t)\#xs) \in$ *cptn-mod* $]\!] \Longrightarrow$
  $(\Gamma,(P,s)\#(Skip,\ t)\#xs) \in$*cptn-mod*

| *CptnModThrow*: $[\![\Gamma\vdash_c(P,s) \rightarrow (Throw,t);$ *redex* $P = P;$
  $(\Gamma,(Throw,\ t)\#xs) \in$ *cptn-mod* $]\!] \Longrightarrow$
  $(\Gamma,(P,s)\#(Throw,\ t)\#xs) \in$*cptn-mod*

| *CptnModCondT*: $[\![(\Gamma,(P0,\ Normal\ s)\#ys) \in$ *cptn-mod*; $s \in b$ $]\!] \Longrightarrow$
  $(\Gamma,((Cond\ b\ P0\ P1),\ Normal\ s)\#(P0,\ Normal\ s)\#ys) \in$ *cptn-mod*
| *CptnModCondF*: $[\![(\Gamma,(P1,\ Normal\ s)\#ys) \in$ *cptn-mod*; $s \notin b$ $]\!] \Longrightarrow$
  $(\Gamma,((Cond\ b\ P0\ P1),\ Normal\ s)\#(P1,\ Normal\ s)\#ys) \in$ *cptn-mod*
| *CptnModSeq1*:
  $[\![(\Gamma,(P0,\ s)\#xs) \in$ *cptn-mod*; zs=*map* (*lift P1*) *xs* $]\!] \Longrightarrow$
  $(\Gamma,((Seq\ P0\ P1),\ s)\#zs) \in$ *cptn-mod*

| *CptnModSeq2*:
$[\![(\Gamma,\ (P0,\ s)\#xs) \in cptn\text{-}mod;\ fst(last\ ((P0,\ s)\#xs)) = Skip;$
$(\Gamma,(P1,\ snd(last\ ((P0,\ s)\#xs)))\#ys) \in cptn\text{-}mod;$
$zs{=}(map\ (lift\ P1)\ xs)@((P1,\ snd(last\ ((P0,\ s)\#xs)))\#ys)\ ]\!] \Longrightarrow$
$(\Gamma,((Seq\ P0\ P1),\ s)\#zs) \in cptn\text{-}mod$

| *CptnModSeq3*:
$[\![(\Gamma,\ (P0,\ Normal\ s)\#xs) \in cptn\text{-}mod;$
$fst(last\ ((P0,\ Normal\ s)\#xs)) = Throw;$
$snd(last\ ((P0,\ Normal\ s)\#xs)) = Normal\ s';$
$(\Gamma,(Throw,Normal\ s')\#ys) \in cptn\text{-}mod;$
$zs{=}(map\ (lift\ P1)\ xs)@((Throw,Normal\ s')\#ys)\ ]\!] \Longrightarrow$
$(\Gamma,((Seq\ P0\ P1),\ Normal\ s)\#zs) \in cptn\text{-}mod$

| *CptnModWhile1*:
$[\![(\Gamma,\ (P,\ Normal\ s)\#xs) \in cptn\text{-}mod;\ s \in b;$
$zs{=}map\ (lift\ (While\ b\ P))\ xs\ ]\!] \Longrightarrow$
$(\Gamma,\ ((While\ b\ P),\ Normal\ s)\#$
$((Seq\ P\ (While\ b\ P)),Normal\ s)\#zs) \in cptn\text{-}mod$

| *CptnModWhile2*:
$[\![\ (\Gamma,\ (P,\ Normal\ s)\#xs) \in cptn\text{-}mod;$
$fst(last\ ((P,\ Normal\ s)\#xs)){=}Skip;\ s \in b;$
$zs{=}(map\ (lift\ (While\ b\ P))\ xs)@$
$(While\ b\ P,\ snd(last\ ((P,\ Normal\ s)\#xs)))\#ys;$
$(\Gamma,(While\ b\ P,\ snd(last\ ((P,\ Normal\ s)\#xs)))\#ys) \in$
$cptn\text{-}mod]\!] \Longrightarrow$
$(\Gamma,(While\ b\ P,\ Normal\ s)\#$
$(Seq\ P\ (While\ b\ P),\ Normal\ s)\#zs) \in cptn\text{-}mod$

| *CptnModWhile3*:
$[\![\ (\Gamma,\ (P,\ Normal\ s)\#xs) \in cptn\text{-}mod;$
$fst(last\ ((P,\ Normal\ s)\#xs)){=}Throw;\ s \in b;$
$snd(last\ ((P,\ Normal\ s)\#xs)) = Normal\ s';$
$(\Gamma,(Throw,Normal\ s')\#ys) \in cptn\text{-}mod;$
$zs{=}(map\ (lift\ (While\ b\ P))\ xs)@((Throw,Normal\ s')\#ys)]\!] \Longrightarrow$
$(\Gamma,(While\ b\ P,\ Normal\ s)\#$
$(Seq\ P\ (While\ b\ P),\ Normal\ s)\#zs) \in cptn\text{-}mod$

| *CptnModCall*: $[\![(\Gamma,(bdy,\ Normal\ s)\#ys) \in cptn\text{-}mod;\Gamma\ p = Some\ bdy;\ bdy{\neq}Call\ p\ ]\!] \Longrightarrow$
$(\Gamma,((Call\ p),\ Normal\ s)\#(bdy,\ Normal\ s)\#ys) \in cptn\text{-}mod$
| *CptnModDynCom*: $[\![(\Gamma,(c\ s,\ Normal\ s)\#ys) \in cptn\text{-}mod\ ]\!] \Longrightarrow$
$(\Gamma,(DynCom\ c,\ Normal\ s)\#(c\ s,\ Normal\ s)\#ys) \in cptn\text{-}mod$

| *CptnModGuard*: $[\![(\Gamma,(c,\ Normal\ s)\#ys) \in cptn\text{-}mod;\ s \in g\ ]\!] \Longrightarrow$
$(\Gamma,(Guard\ f\ g\ c,\ Normal\ s)\#(c,\ Normal\ s)\#ys) \in cptn\text{-}mod$

| *CptnModCatch1*: $[\![(\Gamma,(P0,\ s)\#xs) \in cptn\text{-}mod;\ zs=map\ (lift\text{-}catch\ P1)\ xs\ ]\!]$
$\Longrightarrow (\Gamma,((Catch\ P0\ P1),\ s)\#zs) \in cptn\text{-}mod$
| *CptnModCatch2*:
$[\![(\Gamma,\ (P0,\ s)\#xs) \in cptn\text{-}mod;\ fst(last\ ((P0,\ s)\#xs)) = Skip;$
$(\Gamma,(Skip,snd(last\ ((P0,\ s)\#xs)))\#ys) \in cptn\text{-}mod;$
$zs=(map\ (lift\text{-}catch\ P1)\ xs)@((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)\ ]\!] \Longrightarrow$
$(\Gamma,((Catch\ P0\ P1),\ s)\#zs) \in cptn\text{-}mod$

| *CptnModCatch3*:
$[\![(\Gamma,\ (P0,\ Normal\ s)\#xs) \in cptn\text{-}mod;\ fst(last\ ((P0,\ Normal\ s)\#xs)) = Throw;$
$snd(last\ ((P0,\ Normal\ s)\#xs)) = Normal\ s';$
$(\Gamma,(P1,\ snd(last\ ((P0,\ Normal\ s)\#xs)))\#ys) \in cptn\text{-}mod;$
$zs=(map\ (lift\text{-}catch\ P1)\ xs)@((P1,\ snd(last\ ((P0,\ Normal\ s)\#xs)))\#ys)\ ]\!] \Longrightarrow$
$(\Gamma,((Catch\ P0\ P1),\ Normal\ s)\#zs) \in cptn\text{-}mod$

lemmas *CptnMod-induct* = *cptn-mod.induct* [*of* - [(*c*,*s*)], *split-format* (*complete*), *case-names*
CptnModOne CptnModEnv CptnModSkip CptnModThrow CptnModCondT CptnModCondF
CptnModSeq1 CptnModSeq2 CptnModSeq3 CptnModSeq4 CptnModWhile1 CptnModWhile2 CptnModWhile3 CptnModCall CptnModDynCom CptnModGuard
CptnModCatch1 CptnModCatch2 CptnModCatch3, *induct set*]

inductive-cases *CptnMod-elim-cases* [*cases set*]:
$(\Gamma,(Skip,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Guard\ f\ g\ c,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Basic\ f\ e,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Spec\ r\ e,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Seq\ c1\ c2,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Cond\ b\ c1\ c2,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Await\ b\ c2\ e,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Call\ p,\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(DynCom\ c,s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Throw,s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Catch\ c1\ c2,s)\#u\#xs) \in cptn\text{-}mod$

inductive-cases *CptnMod-Normal-elim-cases* [*cases set*]:
$(\Gamma,(Skip,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Guard\ f\ g\ c,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Basic\ f\ e,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Spec\ r\ e,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Seq\ c1\ c2,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Cond\ b\ c1\ c2,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Await\ b\ c2\ e,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Call\ p,\ Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(DynCom\ c,Normal\ s)\#u\#xs) \in cptn\text{-}mod$
$(\Gamma,(Throw,Normal\ s)\#u\#xs) \in cptn\text{-}mod$

(Γ,(*Catch c1 c2*,*Normal s*)#*u*#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Normal s*)#(*P*,*s'*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Abrupt s*)#(*P*,*Abrupt s'*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Stuck*)#(*P*,*Stuck*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Fault f*)#(*P*,*Fault f*)#*xs*) ∈ *cptn-mod*

inductive-cases *CptnMod-env-elim-cases* [*cases set*]:
(Γ,(*P*,*Normal s*)#(*P*,*s'*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Abrupt s*)#(*P*,*Abrupt s'*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Stuck*)#(*P*,*Stuck*)#*xs*) ∈ *cptn-mod*
(Γ,(*P*,*Fault f*)#(*P*,*Fault f*)#*xs*) ∈ *cptn-mod*

8.7 Equivalence of small semantics and computational

lemma *last-length*: ((*a*#*xs*)!(*length xs*))=*last* (*a*#*xs*)
 by (*induct xs*) *auto*

definition *catch-cond*
where
catch-cond zs Q xs P s s'' s' Γ ≡ (*zs*=(*map* (*lift-catch Q*) *xs*) ∨
 ((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
 snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* ∧ *s*=*Normal s''*∧
 (∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
 zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*))))
∨
 ((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
 (∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod* ∧
 zs=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*)))))

lemma *div-catch*: **assumes** *cptn-m*:(Γ,*list*) ∈ *cptn-mod*
shows (∀ *s P Q zs*. *list*=(*Catch P Q*, *s*)#*zs* ⟶
 (∃ *xs s' s''*.
 (Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod* ∧
 catch-cond zs Q xs P s s'' s' Γ))

unfolding *catch-cond-def*
using *cptn-m*
proof (*induct rule*: *cptn-mod.induct*)
case (*CptnModOne* Γ *P s*)
 thus *?case* **using** *cptn-mod.CptnModOne* **by** *blast*
next
 case (*CptnModSkip* Γ *P s t xs*)
 from *CptnModSkip.hyps*
 have *step*: $\Gamma\vdash_c$ (*P*, *s*) → (*Skip*, *t*) **by** *auto*
 from *CptnModSkip.hyps*
 have *noskip*: ~(*P*=*Skip*) **using** *stepc-elim-cases*(*1*) **by** *blast*
 have *no-catch*: ∀ *p1 p2*. ¬(*P*=*Catch p1 p2*) **using** *CptnModSkip.hyps*(*2*) *redex-not-Catch*
by *auto*

  from *CptnModSkip.hyps*
  have *in-cptn-mod*: (Γ, (*Skip*, *t*) # *xs*) ∈ *cptn-mod* **by** *auto*
  then show *?case* **using** *no-catch* **by** *simp*
next
  case (*CptnModThrow* Γ *P s t xs*)
  from *CptnModThrow.hyps*
  have *step*: $\Gamma\vdash_c$ (*P*, *s*) → (*Throw*, *t*) **by** *auto*
  from *CptnModThrow.hyps*
  have *in-cptn-mod*: (Γ, (*Throw*, *t*) # *xs*) ∈ *cptn-mod* **by** *auto*
 have *no-catch*: ∀ *p1 p2*. ¬(*P*=*Catch p1 p2*) **using** *CptnModThrow.hyps*(*2*) *redex-not-Catch*
by *auto*
  then show *?case* **by** *auto*
next
  case (*CptnModCondT* Γ *P0 s ys b P1*)
  thus *?case* **using** *CptnModOne* **by** *blast*
next
  case (*CptnModCondF* Γ *P0 s ys b P1*)
  thus *?case* **using** *CptnModOne* **by** *blast*
next
  case (*CptnModCatch1 sa P Q zs*)
  thus *?case* **by** *blast*
next
  case (*CptnModCatch2* Γ *P0 s xs ys zs P1*)
  from *CptnModCatch2.hyps*(*3*)
  have *last*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *Skip*
     by (*simp add*: *last-length*)
  have *P0cptn*:(Γ, (*P0*, *s*) # *xs*) ∈ *cptn-mod* **by** *fact*
  then have *zs* = *map* (*lift-catch P1*) *xs* @((*Skip*,*snd*(*last* ((*P0*, *s*)#*xs*)))#*ys*) **by**
(*simp add*:*CptnModCatch2.hyps*)
  show *?case*
  proof −{
    fix *sa P Q zsa*
    assume *eq*:(*Catch P0 P1*, *s*) # *zs* = (*Catch P Q*, *sa*) # *zsa*
    then have *P0* =*P* ∧ *P1* = *Q* ∧ *s*=*sa* ∧ *zs*=*zsa* **by** *auto*
    then have (*P0*, *s*) = (*P*, *sa*) **by** *auto*
    have *last* ((*P0*, *s*) # *xs*) = ((*P*, *sa*) # *xs*) ! *length xs*
      by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› *last-length*)
    then have *zs* = (*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P0*, *s*)#*xs*)))#*ys*)
      using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift-catch P1*)
xs @ ((*Skip*,*snd*(*last* ((*P0*, *s*)#*xs*)))#*ys*)›
      by *force*
    then have (∃ *xs s′ s″*. ((Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod* ∧
          ((*zs*=(*map* (*lift-catch Q*) *xs*) ∨
          ((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
            snd(*last* ((*P*, *s*)#*xs*)) = *Normal s′* ∧ *s*=*Normal s″*∧
            (∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
             zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*))))
∨
            (∃ *ys*. ((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧ (Γ,(*Skip*,*snd*(*last* ((*P*,

s)#*xs*)))#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P0*, *s*)#*xs*)))#*ys*))))))))
using *P0cptn* ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› *last* *CptnModCatch2.hyps*(*4*) **by** *blast*
}
thus *?thesis* **by** *auto*
qed
next
case (*CptnModCatch3* Γ *P0 s xs s′ P1 ys zs*)
from *CptnModCatch3.hyps*(*3*)
have *last*:*fst* (((*P0*, *Normal s*) # *xs*) ! *length xs*) = *Throw*
by (*simp add*: *last-length*)
from *CptnModCatch3.hyps*(*4*)
have *lastnormal*:*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s′*
by (*simp add*: *last-length*)
have *P0cptn*:(Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod* **by** *fact*
from *CptnModCatch3.hyps*(*5*) **have** *P1cptn*:(Γ, (*P1*, *snd* (((*P0*, *Normal s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod*
by (*simp add*: *last-length*)
then have *zs* = *map* (*lift-catch P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *ys* **by** (*simp add*:*CptnModCatch3.hyps*)
show *?case*
proof −**{**
fix *sa P Q zsa*
assume *eq*:(*Catch P0 P1*, *Normal s*) # *zs* = (*Catch P Q*, *Normal sa*) # *zsa*
then have *P0* =*P* ∧ *P1* = *Q* ∧ *Normal s*= *Normal sa* ∧ *zs*=*zsa* **by** *auto*
have *last* ((*P0*, *Normal s*) # *xs*) = ((*P*, *Normal sa*) # *xs*) ! *length xs*
by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last-length*)
then have *zsa* = *map* (*lift-catch Q*) *xs* @ (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift-catch P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *ys*› **by** *force*
then have (Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod* ∧ (*fst*(((*P*, *Normal s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *Normal s*)#*xs*)) = *Normal s′* ∧
(∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *Normal s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *Normal s*)#*xs*)!*length xs*))#*ys*)))
using *lastnormal P1cptn P0cptn* ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last*
by *auto*
}note *this* [*of P0 P1 s zs*] **thus** *?thesis* **by** *blast* **qed**
next
case (*CptnModEnv* Γ *P s t xs*)
then have *step*:(Γ, (*P*, *t*) # *xs*) ∈ *cptn-mod* **by** *auto*
have *step-e*: $\Gamma\vdash_c$ (*P*, *s*) $\rightarrow_e$ (*P*, *t*) **using** *CptnModEnv* **by** *auto*
show *?case*
proof (*cases P*)

case (*Catch P1 P2*)
then have *eq-P-Catch*:(*P*, *t*) # *xs* = (*LanguageCon.com.Catch P1 P2*, *t*) # *xs* **by** *auto*
then obtain *xsa* t' t'' **where**
p1:(Γ, (*P1*, *t*) # *xsa*) ∈ *cptn-mod* **and** *p2*:
(*xs* = *map* (*lift-catch P2*) *xsa* ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal* t' ∧
t = *Normal* t'' ∧
(∃ *ys*. (Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod* ∧
xs =
map (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*.(Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*)))
using *CptnModEnv*(*3*) **by** *auto*
have *all-step*:(Γ, (*P1*, *s*)#((*P1*, *t*) # *xsa*)) ∈ *cptn-mod*
by (*metis p1 Env Env-n cptn-mod.CptnModEnv env-normal-s step-e*)
show *?thesis* **using** *p2*
proof
assume *xs* = *map* (*lift-catch P2*) *xsa*
have (*P*, *t*) # *xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*)
by (*simp add*: ‹*xs* = *map* (*lift-catch P2*) *xsa*› *lift-catch-def local.Catch*)
thus *?thesis* **using** *all-step eq-P-Catch* **by** *fastforce*
next
assume
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal* t' ∧
t = *Normal* t'' ∧
(∃ *ys*. (Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod* ∧
xs =
map (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*))
then show *?thesis*
proof
assume
a1:*fst* (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal* t' ∧
t = *Normal* t'' ∧
(∃ *ys*. (Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*)
then obtain *ys* **where** *p2-exec*:(Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length*

xsa)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*
by *fastforce*
from *a1* **obtain** *t1* **where** *t-normal*: *t*=*Normal t1*
using *env-normal-s′-normal-s* **by** *blast*
have *f1*:*fst* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *t*)#*xsa*)) = *LanguageCon.com.Throw*
using *a1* **by** *fastforce*
from *a1* **have** *last-normal*: *snd* (*last* ((*P1*, *s*)#(*P1*, *t*) # *xsa*)) = *Normal t′*
by *fastforce*
then have *p2-long-exec*: (Γ, (*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *s*)#*xsa*))) # *ys*) ∈ *cptn-mod* ∧
(*P*, *t*)#*xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @
(*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *s*)#*xsa*))) # *ys* **using** *p2-exec*
by (*simp add*: *lift-catch-def local.Catch*)
thus *?thesis* **using** *a1 f1 last-normal all-step eq-P-Catch*
by (*clarify*, *metis* (*no-types*) *list.size*(*4*) *not-step-c-env step-e*)
next
assume
as1:*fst* (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*))
then obtain *ys* **where** *p1*:(Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod* ∧
(*P*, *t*)#*xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*)
proof −
assume *a1*: ⋀*ys*. (Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*))) # *ys*) ∈ *cptn-mod* ∧ (*P*, *t*) # *xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @ (*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*))) # *ys* ⟹ *thesis*
have (*LanguageCon.com.Catch P1 P2*, *t*) # *map* (*lift-catch P2*) *xsa* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*)
by (*simp add*: *lift-catch-def*)
thus *?thesis*
using *a1 as1 eq-P-Catch* **by** *moura*
qed
from *as1* **have** *p2*: *fst* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *t*) #*xsa*)) = *LanguageCon.com.Skip*
by *fastforce*
thus *?thesis* **using** *p1 all-step eq-P-Catch* **by** *fastforce*
qed
qed
qed (*auto*)
qed(*force*+)

definition *seq-cond*
where
seq-cond zs Q xs P s s'' s' $\Gamma \equiv$ (*zs*=(*map* (*lift Q*) *xs*) $\vee$
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* $\wedge$
($\exists$ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) $\in$ *cptn-mod* $\wedge$
zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*)))) $\vee$
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* $\wedge$
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* $\wedge$ *s*=*Normal s''*$\wedge$
($\exists$ *ys*. (Γ,(*Throw*,*Normal s'*)#*ys*) $\in$ *cptn-mod* $\wedge$
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*)))))

lemma *div-seq*: **assumes** *cptn-m*:(Γ,*list*) $\in$ *cptn-mod*
shows ($\forall$ *s P Q zs*. *list*=(*Seq P Q*, *s*)#*zs* $\longrightarrow$
($\exists$ *xs s' s''*.
(Γ,(*P*, *s*)#*xs*) $\in$ *cptn-mod* $\wedge$
seq-cond zs Q xs P s s'' s' Γ))

unfolding *seq-cond-def*
using *cptn-m*
proof (*induct rule*: *cptn-mod.induct*)
case (*CptnModOne* Γ *P s*)
thus *?case* **using** *cptn-mod.CptnModOne* **by** *blast*
next
case (*CptnModSkip* Γ *P s t xs*)
from *CptnModSkip.hyps*
have *step*: $\Gamma \vdash_c$ (*P*, *s*) $\rightarrow$ (*Skip*, *t*) **by** *auto*
from *CptnModSkip.hyps*
have *noskip*: ~(*P*=*Skip*) **using** *stepc-elim-cases*(*1*) **by** *blast*
have *x*: $\forall$ *c c1 c2*. *redex c* = *Seq c1 c2* $\Longrightarrow$ *False*
using *redex-not-Seq* **by** *blast*
from *CptnModSkip.hyps*
have *in-cptn-mod*: (Γ, (*Skip*, *t*) # *xs*) $\in$ *cptn-mod* **by** *auto*
then show *?case* **using** *CptnModSkip.hyps*(*2*) *SmallStepCon.redex-not-Seq* **by** *blast*
next
case (*CptnModThrow* Γ *P s t xs*)
from *CptnModThrow.hyps*
have *step*: $\Gamma \vdash_c$ (*P*, *s*) $\rightarrow$ (*Throw*, *t*) **by** *auto*
moreover from *CptnModThrow.hyps*
have *in-cptn-mod*: (Γ, (*Throw*, *t*) # *xs*) $\in$ *cptn-mod* **by** *auto*
have *no-seq*: $\forall$ *p1 p2*. $\neg$(*P*=*Seq p1 p2*) **using** *CptnModThrow.hyps*(*2*) *redex-not-Seq* **by** *auto*
ultimately show *?case* **by** *auto*
next
case (*CptnModCondT* Γ *P0 s ys b P1*)
thus *?case* **by** *auto*
next

case (*CptnModCondF* Γ *P0 s ys b P1*)
thus *?case* **by** *auto*
next
case (*CptnModSeq1* Γ *P0 s xs zs P1*)
thus *?case* **by** *blast*
next
case (*CptnModSeq2* Γ *P0 s xs P1 ys zs*)
from *CptnModSeq2.hyps*(*3*) *last-length* **have** *last*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *Skip*
by (*simp add*: *last-length*)
have *P0cptn*:(Γ, (*P0*, *s*) # *xs*) ∈ *cptn-mod* **by** *fact*
from *CptnModSeq2.hyps*(*4*) **have** *P1cptn*:(Γ, (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod*
by (*simp add*: *last-length*)
then have *zs* = *map* (*lift P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys* **by** (*simp add*:*CptnModSeq2.hyps*)
show *?case*
proof −{
fix *sa P Q zsa*
assume *eq*:(*Seq P0 P1*, *s*) # *zs* = (*Seq P Q*, *sa*) # *zsa*
then have *P0* =*P* ∧ *P1* = *Q* ∧ *s*=*sa* ∧ *zs*=*zsa* **by** *auto*
have *last* ((*P0*, *s*) # *xs*) = ((*P*, *sa*) # *xs*) ! *length xs*
by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› *last-length*)
then have *zsa* = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length xs*)) # *ys*
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*›
by *force*
then have (∃ *xs s′ s″*. (Γ, (*P*, *sa*) # *xs*) ∈ *cptn-mod* ∧
(*zsa* = *map* (*lift Q*) *xs* ∨
fst (((*P*, *sa*) # *xs*) ! *length xs*) = *Skip* ∧
(∃ *ys*. (Γ, (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod* ∧
zsa = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length xs*)) # *ys*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal s′* ∧ *s*=*Normal s″*∧
(∃ *ys*. (Γ,(*Throw*,*Normal s′*)#*ys*) ∈ *cptn-mod* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s′*)#*ys*))))))

using *P0cptn P1cptn* ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› *last*
by *blast*
}
thus *?case* **by** *auto* **qed**
next
case (*CptnModSeq3* Γ *P0 s xs s′ ys zs P1*)
from *CptnModSeq3.hyps*(*3*)
have *last*:*fst* (((*P0*, *Normal s*) # *xs*) ! *length xs*) = *Throw*
by (*simp add*: *last-length*)
have *P0cptn*:(Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod* **by** *fact*

from *CptnModSeq3.hyps*(*4*)
have *lastnormal*:*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s'*
by (*simp add*: *last-length*)
then have *zs* = *map* (*lift P1*) *xs* @ ((*Throw*, *Normal s'*)#*ys*) **by** (*simp add*:*CptnModSeq3.hyps*)
show *?case*
proof −{
fix *sa P Q zsa*
assume *eq*:(*Seq P0 P1*, *Normal s*) # *zs* = (*Seq P Q*, *Normal sa*) # *zsa*
then have *P0* =*P* ∧ *P1* = *Q* ∧ *Normal s*=*Normal sa* ∧ *zs*=*zsa* **by** *auto*
then have (*P0*, *Normal s*) = (*P*, *Normal sa*) **by** *auto*
have *last* ((*P0*, *Normal s*) # *xs*) = ((*P*, *Normal sa*) # *xs*) ! *length xs*
by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last-length*)
then have *zsa*:*zsa* = (*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*)
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift P1*) *xs* @ ((*Throw*, *Normal s'*)#*ys*)›
by *force*
then have *a1*:(Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod* **using** *CptnModSeq3.hyps*(*5*) **by** *blast*
have (*P*, *Normal sa*::('*b*, '*c*) *xstate*) = (*P0*, *Normal s*)
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› **by** *auto*
then have (∃ *xs s'*. (Γ, (*P*, *Normal sa*) # *xs*) ∈ *cptn-mod* ∧
(*zsa* = *map* (*lift Q*) *xs* ∨
fst (((*P*,*Normal sa*) # *xs*) ! *length xs*) = *Skip* ∧
(∃ *ys*. (Γ, (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod* ∧
zsa = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*) ∨
((*fst*(((*P*, *Normal sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *Normal sa*)#*xs*)) = *Normal s'* ∧
(∃ *ys*. (Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*))))))
using *P0cptn zsa a1 last lastnormal*
by *blast*
}
thus *?thesis* **by** *auto* **qed**
next
case (*CptnModEnv* Γ *P s t zs*)
then have *step*:(Γ, (*P*, *t*) # *zs*) ∈ *cptn-mod* **by** *auto*
have *step-e*: $\Gamma\vdash_c$ (*P*, *s*) $\rightarrow_e$ (*P*, *t*) **using** *CptnModEnv* **by** *auto*
show *?case*
proof (*cases P*)
case (*Seq P1 P2*)
then have *eq-P*:(*P*, *t*) # *zs* = (*LanguageCon.com.Seq P1 P2*, *t*) # *zs* **by** *auto*
then obtain *xs t' t''* **where**
p1:(Γ, (*P1*, *t*) # *xs*) ∈ *cptn-mod* **and** *p2*:
(*zs* = *map* (*lift P2*) *xs* ∨
fst (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧

$(\exists ys.\ (\Gamma,\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod \wedge$
$zs =$
$map\ (lift\ P2)\ xs\ @$
$(P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \vee$
$fst\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P1,\ t)\ \#\ xs)) = Normal\ t' \wedge$
$t = Normal\ t'' \wedge (\exists ys.\ (\Gamma,(Throw,Normal\ t')\#ys) \in cptn\text{-}mod \wedge$
$zs =$
$map\ (lift\ P2)\ xs\ @$
$((LanguageCon.com.Throw,\ Normal\ t')\#ys)))$
using *CptnModEnv(3)* **by** *auto*
have *all-step*:$(\Gamma,\ (P1,\ s)\#((P1,\ t)\ \#\ xs)) \in cptn\text{-}mod$
by (*metis p1 Env Env-n cptn-mod.CptnModEnv env-normal-s step-e*)
show *?thesis* **using** *p2*
proof
assume $zs = map\ (lift\ P2)\ xs$
have $(P,\ t)\ \#\ zs = map\ (lift\ P2)\ ((P1,\ t)\ \#\ xs)$
by (*simp add*: ‹$zs = map\ (lift\ P2)\ xs$› *lift-def local.Seq*)
thus *?thesis* **using** *all-step eq-P* **by** *fastforce*
next
assume
$fst\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Skip \wedge$
$(\exists ys.\ (\Gamma,\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod \wedge$
$zs = map\ (lift\ P2)\ xs\ @\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \vee$
$fst\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P1,\ t)\ \#\ xs)) = Normal\ t' \wedge$
$t = Normal\ t'' \wedge (\exists ys.\ (\Gamma,(Throw,Normal\ t')\#ys) \in cptn\text{-}mod \wedge$
$zs = map\ (lift\ P2)\ xs\ @\ ((LanguageCon.com.Throw,\ Normal\ t')\#ys))$
then show *?thesis*
proof
assume
a1:$fst\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Skip \wedge$
$(\exists ys.\ (\Gamma,\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod \wedge$
$zs = map\ (lift\ P2)\ xs\ @\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys)$
from *a1* **obtain** *ys* **where**
p2-exec:$(\Gamma,\ (P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod$
$\wedge$
$zs = map\ (lift\ P2)\ xs\ @$
$(P2,\ snd\ (((P1,\ t)\ \#\ xs)\ !\ length\ xs))\ \#\ ys$
by *auto*
have *f1*:$fst\ (((P1,\ s)\#(P1,\ t)\ \#\ xs)\ !\ length\ ((P1,\ t)\#xs)) =$
$LanguageCon.com.Skip$
using *a1* **by** *fastforce*
then have *p2-long-exec*:
$(\Gamma,\ (P2,\ snd\ (((P1,\ s)\#(P1,\ t)\ \#\ xs)\ !\ length\ ((P1,\ t)\#xs)))\ \#\ ys)$
$\in cptn\text{-}mod \wedge$
$(P,\ t)\#zs = map\ (lift\ P2)\ ((P1,\ t)\ \#\ xs)\ @$
$(P2,\ snd\ (((P1,\ s)\#(P1,\ t)\ \#\ xs)\ !\ length\ ((P1,\ t)\#xs)))\ \#\ ys$
using *p2-exec* **by** (*simp add*: *lift-def local.Seq*)

thus *?thesis* **using** *a1 f1 all-step eq-P* **by** *blast*
next
assume
a1:*fst* (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xs*)) = *Normal t*′ ∧ *t* = *Normal t*′′ ∧
(∃ *ys*. (Γ,(*Throw*,*Normal t*′)#*ys*) ∈ *cptn-mod* ∧
zs = *map* (*lift P2*) *xs* @ ((*LanguageCon.com.Throw*, *Normal t*′)#*ys*))

then have *last-throw*:
fst (((*P1*, *s*)#(*P1*, *t*) # *xs*) ! *length* ((*P1*, *t*) #*xs*)) = *Language-Con.com.Throw*
by *fastforce*
from *a1* **have** *last-normal*: *snd* (*last* ((*P1*, *s*)#(*P1*, *t*) # *xs*)) = *Normal t*′
by *fastforce*
have *seq-lift*:
(*LanguageCon.com.Seq P1 P2*, *t*) # *map* (*lift P2*) *xs* = *map* (*lift P2*) ((*P1*, *t*) # *xs*)
by (*simp add*: *a1 lift-def*)
thus *?thesis* **using** *a1 last-throw last-normal all-step eq-P*
by (*clarify*, *metis* (*no-types*, *lifting*) *append-Cons env-normal-s*′*-normal-s step-e*)
qed
qed
qed (*auto*)
qed (*force*)+

lemma *cptn-onlyif-cptn-mod-aux*:
assumes *stepseq*:$\Gamma\vdash_c$ (*P*, *s*) → (*Q*,*t*) **and**
stepmod:(Γ,(*Q*,*t*)#*xs*) ∈ *cptn-mod*
shows (Γ,(*P*,*s*)#(*Q*,*t*)#*xs*) ∈ *cptn-mod*
using *stepseq stepmod*
proof (*induct arbitrary*: *xs*)
case (*Basicc f s*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.Basicc*)
next
case (*Specc s t r*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.Specc*)
next
case (*SpecStuckc s r*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.SpecStuckc*)
next
case (*Guardc s g f c*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModGuard*)
next
case (*GuardFaultc*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.GuardFaultc*)
next

case (*Seqc c1 s c1′ s′ c2*)
have *step*: $\Gamma\vdash_c$ (*c1*, *s*) → (*c1′*, *s′*) **by** (*simp add*: *Seqc.hyps*(*1*))
then have *nsc1*: *c1*≠*Skip* **using** *stepc-elim-cases*(*1*) **by** *blast*
have *assum*: (Γ, (*Seq c1′ c2*, *s′*) # *xs*) ∈ *cptn-mod* **using** *Seqc.prems* **by** *blast*
have *divseq*:(∀ *s P Q zs*. (*Seq c1′ c2*, *s′*) # *xs*=(*Seq P Q*, *s*)#*zs* ⟶
(∃ *xs sv′ sv″*. ((Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod* ∧
(*zs*=(*map* (*lift Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod*
∧
zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length*
xs))#*ys*)))) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal sv′* ∧ *s′*=*Normal sv″*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sv′*)#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal sv′*)#*ys*))
))))

)) **using** *div-seq* [*OF assum*] **unfolding** *seq-cond-def* **by** *auto*
{**fix** *sa P Q zsa*
assume *ass*:(*Seq c1′ c2*, *s′*) # *xs* = (*Seq P Q*, *sa*) # *zsa*
then have *eqs*:*c1′* = *P* ∧ *c2* = *Q* ∧ *s′* = *sa* ∧ *xs* = *zsa* **by** *auto*
then have (∃ *xs sv′ sv″*. (Γ, (*P*, *sa*) # *xs*) ∈ *cptn-mod* ∧
(*zsa* = *map* (*lift Q*) *xs* ∨
fst (((*P*, *sa*) # *xs*) ! *length xs*) = *Skip* ∧
(∃ *ys*. (Γ, (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length xs*)) # *ys*) ∈
cptn-mod ∧
zsa = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length*
xs)) # *ys*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal sv′* ∧ *s′*=*Normal sv″*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sv′*)#*ys*) ∈ *cptn-mod* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal sv′*)#*ys*))))))
using *ass divseq* **by** *blast*
} **note** *conc*=*this* [*of c1′ c2 s′ xs*]
then obtain *xs′ sa′ sa″*
where *split*:(Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod* ∧
(*xs* = *map* (*lift c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈
cptn-mod ∧
xs = *map* (*lift c2*) *xs′* @ (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length*
xs′)) # *ys*) ∨
((*fst*(((*c1′*, *s′*)#*xs′*)!*length xs′*)=*Throw* ∧
snd(*last* ((*c1′*, *s′*)#*xs′*)) = *Normal sa′* ∧ *s′*=*Normal sa″*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sa′*)#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift c2*) *xs′*)@((*Throw*,*Normal sa′*)#*ys*))
))) **by** *blast*
then have (*xs* = *map* (*lift c2*) *xs′* ∨

fst (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈
cptn-mod ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length*
xs')) # *ys*) ∨
((*fst*(((*c1'*, *s'*)#*xs'*)!*length xs'*)=*Throw* ∧
snd(*last* ((*c1'*, *s'*)#*xs'*)) = *Normal sa'* ∧ *s'*=*Normal sa''*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sa'*)#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift c2*) *xs'*)@((*Throw*,*Normal sa'*)#*ys*))))) **by**
auto
thus *?case*
proof{
assume *c1'nonf*:*xs* = *map* (*lift c2*) *xs'*
then have *c1'cptn*:(Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod* **using** *split* **by** *blast*
then have *induct-step*: (Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod*
using *Seqc.hyps*(*2*) **by** *blast*
then have (*Seq c1' c2*, *s'*)#*xs* = *map* (*lift c2*) ((*c1'*, *s'*)#*xs'*)
using *c1'nonf*
by (*simp add*: *CptnModSeq1 lift-def*)
thus *?thesis*
using *c1'nonf c1'cptn induct-step* **by** (*auto simp add*: *CptnModSeq1*)
next
assume *fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) #
ys) ∨
((*fst*(((*c1'*, *s'*)#*xs'*)!*length xs'*)=*Throw* ∧
snd(*last* ((*c1'*, *s'*)#*xs'*)) = *Normal sa'* ∧ *s'*=*Normal sa''*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sa'*)#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift c2*) *xs'*)@((*Throw*,*Normal sa'*)#*ys*))))
thus *?thesis*
proof
assume *assth*:*fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) #
ys)
then obtain *ys*
where *split'*:(Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈
cptn-mod ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) #
ys
by *auto*
then have *c1'cptn*:(Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod* **using** *split* **by** *blast*
then have *induct-step*: (Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod*
using *Seqc.hyps*(*2*) **by** *blast*
then have *seqmap*:(*Seq c1 c2*, *s*)#(*Seq c1' c2*, *s'*)#*xs* = *map* (*lift c2*)
((*c1*,*s*)#(*c1'*, *s'*)#*xs'*) @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*
using *split'*
by (*simp add*: *CptnModSeq2 lift-def*)

then have *lastc1*:*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*) = ((*c1'*, *s'*) # *xs'*) ! *length xs'*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Skip*
using *assth* **by** *fastforce*
thus *?thesis*
using *seqmap split' last-length cptn-mod.CptnModSeq2*
induct-step lastc1 lastc1skip
by *fastforce*
next
assume *assm*:((*fst*(((*c1'*, *s'*)#*xs'*)!*length xs'*)=*Throw* ∧
snd(*last* ((*c1'*, *s'*)#*xs'*)) = *Normal sa'* ∧ *s'*=*Normal sa''*∧
(∃ *ys*. (Γ,(*Throw*,*Normal sa'*)#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift c2*) *xs'*)@((*Throw*,*Normal sa'*)#*ys*))))
then have *s'eqsa''*: *s'*=*Normal sa''* **by** *auto*
then have *snormal*: ∃ *ns*. *s*=*Normal ns* **by** (*metis Seqc.hyps*(*1*) *step-Abrupt-prop step-Fault-prop step-Stuck-prop xstate.exhaust*)
then have *c1'cptn*:(Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod* **using** *split* **by** *blast*

then have *induct-step*: (Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod*
using *Seqc.hyps*(*2*) **by** *blast*
then obtain *ys* **where** *seqmap*:(*Seq c1' c2*, *s'*)#*xs* = (*map* (*lift c2*) ((*c1'*, *s'*)#*xs'*))@((*Throw*,*Normal sa'*)#*ys*)
using *assm*
proof −
assume *a1*: ⋀*ys*. (*LanguageCon.com.Seq c1' c2*, *s'*) # *xs* = *map* (*lift c2*) ((*c1'*, *s'*) # *xs'*) @ (*LanguageCon.com.Throw*, *Normal sa'*) # *ys* ⟹ *thesis*
have (*LanguageCon.com.Seq c1' c2*, *Normal sa''*) # *map* (*lift c2*) *xs'* = *map* (*lift c2*) ((*c1'*, *s'*) # *xs'*)
by (*simp add*: *assm lift-def*)
thus *?thesis*
using *a1 assm* **by** *moura*
qed
then have *lastc1*:*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*) = ((*c1'*, *s'*) # *xs'*) ! *length xs'*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Throw*
using *assm* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Normal sa'*
using *assm* **by** *force*
thus *?thesis*
using *assm c1'cptn induct-step lastc1skip snormal seqmap s'eqsa''*
by (*auto simp add*:*cptn-mod.CptnModSeq3*)
qed
}**qed**

next
case (*SeqSkipc c2 s xs*)
have *c2incptn*:(Γ, (*c2*, *s*) # *xs*) ∈ *cptn-mod* **by** *fact*

then have *1*:(Γ, [(*Skip*, *s*)]) ∈ *cptn-mod* **by** (*simp add*: *cptn-mod.CptnModOne*)
then have *2*:*fst*(*last* ([(*Skip*, *s*)])) = *Skip* **by** *fastforce*
then have *3*:(Γ,(*c2*, *snd*(*last* [(*Skip*, *s*)]))#*xs*) ∈ *cptn-mod*
using *c2incptn* **by** *auto*
then have (*c2*,*s*)#*xs*=(*map* (*lift c2*) [])@(*c2*, *snd*(*last* [(*Skip*, *s*)]))#*xs*
by (*auto simp add*:*lift-def*)
thus *?case* **using** *1 2 3* **by** (*simp add*: *CptnModSeq2*)
next
case (*SeqThrowc c2 s xs*)
have (Γ, [(*Throw*, *Normal s*)]) ∈ *cptn-mod* **by** (*simp add*: *cptn-mod.CptnModOne*)

then obtain *ys* **where** *ys-nil*:*ys*=[] **and** *last*:(Γ, (*Throw*, *Normal s*)#*ys*)∈ *cptn-mod*
by *auto*
moreover have *fst* (*last* ((*Throw*, *Normal s*)#*ys*)) = *Throw* **using** *ys-nil last* **by** *auto*
moreover have *snd* (*last* ((*Throw*, *Normal s*)#*ys*)) = *Normal s* **using** *ys-nil last* **by** *auto*
moreover from *ys-nil* **have** (*map* (*lift c2*) *ys*) = [] **by** *auto*
ultimately show *?case* **using** *SeqThrowc.prems cptn-mod.CptnModSeq3* **by** *fastforce*
next
case (*CondTruec s b c1 c2*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModCondT*)
next
case (*CondFalsec s b c1 c2*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModCondF*)
next
case (*WhileTruec s1 b c*)
have *sinb*: *s1*∈*b* **by** *fact*
have *SeqcWhile*: (Γ, (*Seq c* (*While b c*), *Normal s1*) # *xs*) ∈ *cptn-mod* **by** *fact*
have *divseq*:(∀ *s P Q zs*. (*Seq c* (*While b c*), *Normal s1*) # *xs*=(*Seq P Q*, *s*)#*zs* ⟶
(∃ *xs s'*. ((Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod* ∧
(*zs*=(*map* (*lift Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*)))) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* ∧
(∃ *ys*. (Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*))))))
)) **using** *div-seq* [*OF SeqcWhile*] **by** (*auto simp add*: *seq-cond-def*)
{fix *sa P Q zsa*
assume *ass*:(*Seq c* (*While b c*), *Normal s1*) # *xs* = (*Seq P Q*, *sa*) # *zsa*
then have *eqs*:*c* = *P* ∧ (*While b c*) = *Q* ∧ *Normal s1* = *sa* ∧ *xs* = *zsa* **by** *auto*

then have $(\exists xs\ s'.\ (\Gamma,\ (P,\ sa)\ \#\ xs) \in cptn\text{-}mod\ \wedge$
$(zsa = map\ (lift\ Q)\ xs\ \vee$
$fst\ (((P,\ sa)\ \#\ xs)\ !\ length\ xs) = Skip\ \wedge$
$(\exists ys.\ (\Gamma,\ (Q,\ snd\ (((P,\ sa)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in$
$cptn\text{-}mod\ \wedge$
$zsa = map\ (lift\ Q)\ xs\ @\ (Q,\ snd\ (((P,\ sa)\ \#\ xs)\ !\ length$
$xs))\ \#\ ys)\ \vee$
$((fst(((P,\ sa)\#xs)!length\ xs)=Throw\ \wedge$
$snd(last\ ((P,\ sa)\#xs)) = Normal\ s'\ \wedge$
$(\exists ys.\ \ (\Gamma,(Throw,Normal\ s')\#ys) \in cptn\text{-}mod\ \wedge$
$zsa=(map\ (lift\ Q)\ xs)@((Throw,Normal\ s')\#ys))$
$))))$
using *ass divseq* **by** *auto*
} **note** *conc=this* [*of c While b c Normal s1 xs*]
then obtain $xs'\ s'$
where $split{:}(\Gamma,\ (c,\ Normal\ s1)\ \#\ xs') \in cptn\text{-}mod\ \wedge$
$(xs = map\ (lift\ (While\ b\ c))\ xs'\ \vee$
$fst\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs') = Skip\ \wedge$
$(\exists ys.\ (\Gamma,\ (While\ b\ c,\ snd\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys)$
$\in cptn\text{-}mod\ \wedge$
$xs =$
$map\ (lift\ (While\ b\ c))\ xs'\ @$
$(While\ b\ c,\ snd\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys)\ \vee$
$fst\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs') = Throw\ \wedge$
$snd\ (last\ ((c,\ Normal\ s1)\ \#\ xs')) = Normal\ s'\ \wedge$
$(\exists ys.\ \ (\Gamma,\ ((Throw,\ Normal\ s')\#ys)) \in cptn\text{-}mod\ \wedge$
$xs = map\ (lift\ (While\ b\ c))\ xs'\ @\ ((Throw,\ Normal\ s')\#ys)))$ **by** *auto*
then have $(xs = map\ (lift\ (While\ b\ c))\ xs'\ \vee$
$fst\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs') = Skip\ \wedge$
$(\exists ys.\ (\Gamma,\ (While\ b\ c,\ snd\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys)$
$\in cptn\text{-}mod\ \wedge$
$xs =$
$map\ (lift\ (While\ b\ c))\ xs'\ @$
$(While\ b\ c,\ snd\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys)\ \vee$
$fst\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs') = Throw\ \wedge$
$snd\ (last\ ((c,\ Normal\ s1)\ \#\ xs')) = Normal\ s'\ \wedge$
$(\exists ys.\ \ (\Gamma,\ ((Throw,\ Normal\ s')\#ys)) \in cptn\text{-}mod\ \wedge$
$xs = map\ (lift\ (While\ b\ c))\ xs'\ @\ ((Throw,\ Normal\ s')\#ys)))$ **..**
thus *?case*
proof{
assume $1{:}xs = map\ (lift\ (While\ b\ c))\ xs'$
have $3{:}(\Gamma,\ (c,\ Normal\ s1)\ \#\ xs') \in cptn\text{-}mod$ **using** *split* **by** *auto*
then show *?thesis* **using** *1 cptn-mod.CptnModWhile1 sinb* **by** *fastforce*
next
assume $fst\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs') = Skip\ \wedge$
$(\exists ys.\ (\Gamma,\ (While\ b\ c,\ snd\ (((c,\ Normal\ s1)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys)$
$\in cptn\text{-}mod\ \wedge$
$xs =$
$map\ (lift\ (While\ b\ c))\ xs'\ @$

```
            (While b c, snd (((c, Normal s1) # xs') ! length xs')) # ys) ∨
        fst (((c, Normal s1) # xs') ! length xs') = Throw ∧
        snd (last ((c, Normal s1) # xs')) = Normal s' ∧
        (∃ ys.  (Γ, ((Throw, Normal s')#ys)) ∈ cptn-mod ∧
        xs = map (lift (While b c)) xs' @ ((Throw, Normal s')#ys))
  thus ?case
  proof
    assume asm:fst (((c, Normal s1) # xs') ! length xs') = Skip ∧
           (∃ ys. (Γ, (While b c, snd (((c, Normal s1) # xs') ! length xs')) # ys)
           ∈ cptn-mod ∧
           xs =
           map (lift (While b c)) xs' @
           (While b c, snd (((c, Normal s1) # xs') ! length xs')) # ys)
    then obtain ys
      where asm':(Γ, (While b c, snd (last ((c, Normal s1) # xs'))) # ys)
                 ∈ cptn-mod
                 ∧ xs = map (lift (While b c)) xs' @
                     (While b c, snd (last ((c, Normal s1) # xs'))) # ys
           by (auto simp add: last-length)
    moreover have 3:(Γ, (c, Normal s1) # xs') ∈ cptn-mod using split by auto
    moreover from asm have fst (last ((c, Normal s1) # xs'))  = Skip
         by (simp add: last-length)
    ultimately show ?case using sinb by (auto simp add:CptnModWhile2)
  next
   assume asm: fst (((c, Normal s1) # xs') ! length xs') = Throw ∧
        snd (last ((c, Normal s1) # xs')) = Normal s' ∧
        (∃ ys.  (Γ, ((Throw, Normal s')#ys)) ∈ cptn-mod ∧
        xs = map (lift (While b c)) xs' @ ((Throw, Normal s')#ys))
    moreover have 3:(Γ, (c, Normal s1) # xs') ∈ cptn-mod using split by auto
    moreover from asm have fst (last ((c, Normal s1) # xs'))  = Throw
         by (simp add: last-length)
    ultimately show ?case using sinb by (auto simp add:CptnModWhile3)
  qed
 }qed
next
 case (WhileFalsec s b c)
 thus ?case by (simp add: cptn-mod.CptnModSkip stepc.WhileFalsec)
next
  case (Awaitc s b c t)
  thus ?case by (simp add: cptn-mod.CptnModSkip stepc.Awaitc)
next
  case (AwaitAbruptc s b c t t')
  thus ?case by (simp add: cptn-mod.CptnModThrow stepc.AwaitAbruptc)
next
  case (Callc p bdy s)
  thus ?case by (simp add: cptn-mod.CptnModCall)
next
  case (CallUndefinedc p s)
  thus ?case by (simp add: cptn-mod.CptnModSkip stepc.CallUndefinedc)
```

next
case (*DynComc c s*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModDynCom*)
next
case (*Catchc c1 s c1′ s′ c2*)
have *step*: $\Gamma\vdash_c$ (*c1*, *s*) → (*c1′*, *s′*) **by** (*simp add*: *Catchc.hyps*(*1*))
then have *nsc1*: *c1*≠*Skip* **using** *stepc-elim-cases*(*1*) **by** *blast*
have *assum*: (Γ, (*Catch c1′ c2*, *s′*) # *xs*) ∈ *cptn-mod*
using *Catchc.prems* **by** *blast*
have *divcatch*:(∀ *s P Q zs*. (*Catch c1′ c2*, *s′*) # *xs*=(*Catch P Q*, *s*)#*zs* ⟶
(∃ *xs s′ s″*. ((Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod* ∧
(*zs*=(*map* (*lift-catch Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s′* ∧ *s*=*Normal s″*∧
(∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*))))
∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod* ∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*))

))))
)) **using** *div-catch* [*OF assum*] **by** (*auto simp add*: *catch-cond-def*)
{**fix** *sa P Q zsa*
assume *ass*:(*Catch c1′ c2*, *s′*) # *xs* = (*Catch P Q*, *sa*) # *zsa*
then have *eqs*:*c1′* = *P* ∧ *c2* = *Q* ∧ *s′* = *sa* ∧ *xs* = *zsa* **by** *auto*
then have (∃ *xs sv′ sv″*. ((Γ,(*P*, *sa*)#*xs*) ∈ *cptn-mod* ∧
(*zsa*=(*map* (*lift-catch Q*) *xs*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal sv′* ∧ *s′*=*Normal sv″*∧
(∃ *ys*. (Γ,(*Q*, *snd*(((*P*, *sa*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod* ∧
zsa=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *sa*)#*xs*)!*length xs*))#*ys*))))
∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*P*, *sa*)#*xs*)))#*ys*) ∈ *cptn-mod* ∧
zsa=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *sa*)#*xs*)))#*ys*))))))
) **using** *ass divcatch* **by** *blast*
} **note** *conc*=*this* [*of c1′ c2 s′ xs*]
then obtain *xs′ sa′ sa″*
where *split*:
(Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod* ∧
(*xs* = *map* (*lift-catch c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Throw* ∧
snd (*last* ((*c1′*, *s′*) # *xs′*)) = *Normal sa′* ∧ *s′* = *Normal sa″* ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch c2*) *xs′* @
(*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*) ∈ *cptn-mod* ∧

xs=(*map* (*lift-catch c2*) *xs*′)@((*Skip*,*snd*(*last* ((*c1*′, *s*′)#*xs*′)))#*ys*)))

by *blast*
then have (*xs* = *map* (*lift-catch c2*) *xs*′ ∨
fst (((*c1*′, *s*′) # *xs*′) ! *length xs*′) = *Throw* ∧
snd (*last* ((*c1*′, *s*′) # *xs*′)) = *Normal sa*′ ∧ *s*′ = *Normal sa*′′ ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod* ∧
xs = *map* (*lift-catch c2*) *xs*′ @
(*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′)) # *ys*) ∨
fst (((*c1*′, *s*′) # *xs*′) ! *length xs*′) = *Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*c1*′, *s*′)#*xs*′)))#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift-catch c2*) *xs*′)@((*Skip*,*snd*(*last* ((*c1*′, *s*′)#*xs*′)))#*ys*)))

by *auto*
thus *?case*
proof{
assume *c1*′*nonf*:*xs* = *map* (*lift-catch c2*) *xs*′
then have *c1*′*cptn*:(Γ, (*c1*′, *s*′) # *xs*′) ∈ *cptn-mod* **using** *split* **by** *blast*
then have *induct-step*: (Γ, (*c1*, *s*) # (*c1*′, *s*′)#*xs*′) ∈ *cptn-mod*
using *Catchc.hyps*(*2*) **by** *blast*
then have (*Catch c1*′ *c2*, *s*′)#*xs* = *map* (*lift-catch c2*) ((*c1*′, *s*′)#*xs*′)
using *c1*′*nonf*
by (*simp add*: *CptnModCatch1 lift-catch-def*)
thus *?thesis*
using *c1*′*nonf c1*′*cptn induct-step* **by** (*auto simp add*: *CptnModCatch1*)
next
assume *fst* (((*c1*′, *s*′) # *xs*′) ! *length xs*′) = *Throw* ∧
snd (*last* ((*c1*′, *s*′) # *xs*′)) = *Normal sa*′ ∧ *s*′ = *Normal sa*′′ ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod* ∧
xs =*map* (*lift-catch c2*) *xs*′ @ (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′))
ys) ∨
fst (((*c1*′, *s*′) # *xs*′) ! *length xs*′) = *Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*c1*′, *s*′)#*xs*′)))#*ys*) ∈ *cptn-mod* ∧
xs=(*map* (*lift-catch c2*) *xs*′)@((*Skip*,*snd*(*last* ((*c1*′, *s*′)#*xs*′)))#*ys*))
thus *?thesis*
proof
assume *assth*:
fst (((*c1*′, *s*′) # *xs*′) ! *length xs*′) = *Throw* ∧
snd (*last* ((*c1*′, *s*′) # *xs*′)) = *Normal sa*′ ∧ *s*′ = *Normal sa*′′ ∧
(∃ *ys*. (Γ, (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod* ∧
xs =*map* (*lift-catch c2*) *xs*′ @ (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′))
ys)
then have *s*′*eqsa*′′: *s*′=*Normal sa*′′ **by** *auto*
then have *snormal*: ∃ *ns*. *s*=*Normal ns* **by** (*metis Catchc.hyps*(*1*)
step-Abrupt-prop step-Fault-prop step-Stuck-prop xstate.exhaust)
then obtain *ys*
where *split*′:(Γ, (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′)) # *ys*) ∈
cptn-mod ∧
xs =*map* (*lift-catch c2*) *xs*′ @ (*c2*, *snd* (((*c1*′, *s*′) # *xs*′) ! *length xs*′))

ys
using *assth* **by** *auto*
then have *c1′cptn*:(Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod*
using *split* **by** *blast*
then have *induct-step*: (Γ, (*c1*, *s*) # (*c1′*, *s′*)#*xs′*) ∈ *cptn-mod*
using *Catchc.hyps*(*2*) **by** *blast*
then have *seqmap*:(*Catch c1 c2*, *s*)#(*Catch c1′ c2*, *s′*)#*xs* = *map* (*lift-catch c2*) ((*c1*,*s*)#(*c1′*, *s′*)#*xs′*) @ (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*
using *split′* **by** (*simp add*: *CptnModCatch3 lift-catch-def*)
then have *lastc1*:*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*) = ((*c1′*, *s′*) # *xs′*) ! *length xs′*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *Throw*
using *assth* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *Normal sa′*
using *assth* **by** *force*
thus *?thesis* **using** *snormal seqmap s′eqsa′′ split′ last-length cptn-mod.CptnModCatch3 induct-step lastc1 lastc1skip*
by *fastforce*
next
assume *assm*: *fst* (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (Γ,(*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*) ∈ *cptn-mod* ∧

xs=(*map* (*lift-catch c2*) *xs′*)@((*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*))
then have *c1′cptn*:(Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod* **using** *split* **by** *blast*
then have *induct-step*: (Γ, (*c1*, *s*) # (*c1′*, *s′*)#*xs′*) ∈ *cptn-mod*
using *Catchc.hyps*(*2*) **by** *blast*
then have *map* (*lift-catch c2*) ((*c1′*, *s′*) # *xs′*) = (*Catch c1′ c2*, *s′*) # *map* (*lift-catch c2*) *xs′*
by (*auto simp add*: *lift-catch-def*)
then obtain *ys*
where *seqmap*:(*Catch c1′ c2*, *s′*)#*xs* = (*map* (*lift-catch c2*) ((*c1′*, *s′*)#*xs′*))@((*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*)
using *assm* **by** *fastforce*
then have *lastc1*:*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*) = ((*c1′*, *s′*) # *xs′*) ! *length xs′*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *Skip*
using *assm* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *snd* (*last* ((*c1′*, *s′*) # *xs′*))
using *assm* **by** *force*
thus *?thesis*
using *assm c1′cptn induct-step lastc1skip seqmap* **by** (*auto simp add*:*cptn-mod.CptnModCatch2*)
qed
}qed
next
case (*CatchThrowc c2 s*)

have *c2incptn*:(Γ, (*c2*, *Normal s*) # *xs*) ∈ *cptn-mod* **by** *fact*
then have *1*:(Γ, [(*Throw*, *Normal s*)]) ∈ *cptn-mod* **by** (*simp add*: *cptn-mod.CptnModOne*)
then have *2*:*fst*(*last* ([(*Throw*, *Normal s*)])) = *Throw* **by** *fastforce*
then have *3*:(Γ,(*c2*, *snd*(*last* [(*Throw*, *Normal s*)]))#*xs*) ∈ *cptn-mod*
using *c2incptn* **by** *auto*
then have (*c2*,*Normal s*)#*xs*=(*map* (*lift c2*) [])@(*c2*, *snd*(*last* [(*Throw*, *Normal s*)]))#*xs*
by (*auto simp add*:*lift-def*)
thus *?case* **using** *1 2 3* **by** (*simp add*: *CptnModCatch3*)
next
case (*CatchSkipc c2 s*)
have (Γ, [(*Skip*, *s*)]) ∈ *cptn-mod* **by** (*simp add*: *cptn-mod.CptnModOne*)
then obtain *ys* **where** *ys-nil*:*ys*=[] **and** *last*:(Γ, (*Skip*, *s*)#*ys*)∈ *cptn-mod*
by *auto*
moreover have *fst* (*last* ((*Skip*, *s*)#*ys*)) = *Skip* **using** *ys-nil last* **by** *auto*
moreover have *snd* (*last* ((*Skip*, *s*)#*ys*)) = *s* **using** *ys-nil last* **by** *auto*
moreover from *ys-nil* **have** (*map* (*lift-catch c2*) *ys*) = [] **by** *auto*
ultimately show *?case* **using** *CatchSkipc.prems* **by** *simp* (*simp add*: *cptn-mod.CptnModCatch2 ys-nil*)
next
case (*FaultPropc c f*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.FaultPropc*)
next
case (*AbruptPropc c f*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.AbruptPropc*)
next
case (*StuckPropc c*)
thus *?case* **by** (*simp add*: *cptn-mod.CptnModSkip stepc.StuckPropc*)
qed

lemma *cptn-onlyif-cptn-mod*:
assumes *cptn-asm*:(Γ,*c*) ∈ *cptn*
shows (Γ,*c*) ∈ *cptn-mod*
using *cptn-asm*
proof (*induct*)
case *CptnOne* **thus** *?case* **by** (*rule CptnModOne*)
next
case (*CptnEnv* Γ *P t xs s*) **thus** *?case* **by** (*simp add*: *cptn-mod.CptnModEnv*)
next
case *CptnComp* **thus** *?case*
by (*simp add*: *cptn-onlyif-cptn-mod-aux*)
qed

lemma *lift-is-cptn*:
assumes *cptn-asm*:(Γ,*c*)∈*cptn*
shows (Γ,*map* (*lift P*) *c*) ∈ *cptn*
using *cptn-asm*
proof (*induct*)
case *CptnOne* **thus** *?case* **using** *cptn.simps* **by** *fastforce*

next
  case (*CptnEnv* Γ *P s t xs*) **thus** *?case*
    by (*cases rule*:*step-e.cases*,
      (*simp add*: *cptn.CptnEnv step-e.Env lift-def*),
      (*simp add*: *cptn.CptnEnv step-e.Env-n lift-def*))
next
  case *CptnComp* **thus** *?case* **by** (*simp add*: *Seqc cptn.CptnComp lift-def*)
qed

lemma *lift-catch-is-cptn*:
assumes *cptn-asm*:(Γ,*c*)∈*cptn*
shows (Γ,*map* (*lift-catch P*) *c*) ∈ *cptn*
using *cptn-asm*
proof (*induct*)
  case *CptnOne* **thus** *?case* **using** *cptn.simps* **by** *fastforce*
next
  case *CptnEnv* **thus** *?case* **by** (*cases rule*:*step-e.cases*,
      (*simp add*: *cptn.CptnEnv step-e.Env lift-catch-def*),
      (*simp add*: *cptn.CptnEnv step-e.Env-n lift-catch-def*))
next
  case *CptnComp* **thus** *?case* **by** (*simp add*: *Catchc cptn.CptnComp lift-catch-def*)
qed

lemma *last-lift*: ⟦*xs*≠[]; *fst*(*xs*!(*length xs* − (*Suc 0*)))=*Q*⟧
 ⟹ *fst*((*map* (*lift P*) *xs*)!(*length* (*map* (*lift P*) *xs*)− (*Suc 0*)))=*Seq Q P*
  by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp add*:*lift-def*)

lemma *last-lift-catch*: ⟦*xs*≠[]; *fst*(*xs*!(*length xs* − (*Suc 0*)))=*Q*⟧
 ⟹ *fst*((*map* (*lift-catch P*) *xs*)!(*length* (*map* (*lift-catch P*) *xs*)− (*Suc 0*)))=*Catch Q P*
  by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp add*:*lift-catch-def*)

lemma *last-fst* [*rule-format*]: *P*((*a*#*x*)!*length x*) ⟶ ¬*P a* ⟶ *P* (*x*!(*length x* − (*Suc 0*)))
  by (*induct x*) *simp-all*

lemma *last-fst-esp*:
 fst(((*P*,*s*)#*xs*)!(*length xs*))=*Skip* ⟹ *P*≠*Skip* ⟹ *fst*(*xs*!(*length xs* − (*Suc 0*)))=*Skip*

apply(*erule last-fst*)
apply *simp*
done

lemma *last-snd*: *xs*≠[] ⟹
  snd(((*map* (*lift P*) *xs*))!(*length* (*map* (*lift P*) *xs*) − (*Suc 0*)))=*snd*(*xs*!(*length xs* − (*Suc 0*)))
  by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp-all add*:*lift-def*)

lemma *last-snd-catch*: $xs \neq [] \Longrightarrow$
$snd(((map\ (lift\text{-}catch\ P)\ xs))!(length\ (map\ (lift\text{-}catch\ P)\ xs) - (Suc\ 0)))=snd(xs!(length\ xs - (Suc\ 0)))$
by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp-all add*:*lift-catch-def*)

lemma *Cons-lift*: ((*Seq P Q*), *s*) # (*map* (*lift Q*) *xs*) = *map* (*lift Q*) ((*P*, *s*) # *xs*)
by (*simp add*:*lift-def*)
thm *last-map eq-snd-iff list.inject list.simps*(*9*) *last-length*
lemma *Cons-lift-catch*: ((*Catch P Q*), *s*) # (*map* (*lift-catch Q*) *xs*) = *map* (*lift-catch Q*) ((*P*, *s*) # *xs*)
by (*simp add*:*lift-catch-def*)

lemma *Cons-lift-append*:
((*Seq P Q*), *s*) # (*map* (*lift Q*) *xs*) @ *ys* = *map* (*lift Q*) ((*P*, *s*) # *xs*)@ *ys*
by (*simp add*:*lift-def*)

lemma *Cons-lift-catch-append*:
((*Catch P Q*), *s*) # (*map* (*lift-catch Q*) *xs*) @ *ys* = *map* (*lift-catch Q*) ((*P*, *s*) # *xs*)@ *ys*
by (*simp add*:*lift-catch-def*)

lemma *lift-nth*: $i<length\ xs \Longrightarrow map\ (lift\ Q)\ xs\ !\ i = lift\ Q\ \ (xs!\ i)$
by (*simp add*:*lift-def*)

lemma *lift-catch-nth*: $i<length\ xs \Longrightarrow map\ (lift\text{-}catch\ Q)\ xs\ !\ i = lift\text{-}catch\ Q\ \ (xs!\ i)$
by (*simp add*:*lift-catch-def*)
thm *list.simps*(*9*) *last-length lift-catch-def Cons-lift-catch*
lemma *snd-lift*: $i< length\ xs \Longrightarrow snd(lift\ Q\ (xs\ !\ i))= snd\ (xs\ !\ i)$
by (*cases xs*!*i*) (*simp add*:*lift-def*)

lemma *snd-lift-catch*: $i< length\ xs \Longrightarrow snd(lift\text{-}catch\ Q\ (xs\ !\ i))= snd\ (xs\ !\ i)$
by (*cases xs*!*i*) (*simp add*:*lift-catch-def*)

lemma *Normal-Normal*:
assumes *p1*:(Γ, (*P*, *Normal s*) # *a* # *as*) $\in$ *cptn* **and**
p2:($\exists$ *sb*. *snd* (*last* ((*P*, *Normal s*) # *a* # *as*)) = *Normal sb*)
shows $\exists$ *sa*. *snd a* = *Normal sa*
proof −
obtain *la1 la2* **where** *last-prod*:*last* ((*P*, *Normal s*)# *a*#*as*) = (*la1*,*la2*) **by** *fastforce*
obtain *a1 a2* **where** *a-prod*:*a*=(*a1*,*a2*) **by** *fastforce*
from *p1* **have** *clos-p-a*:$\Gamma\vdash_c (P,Normal\ s) \rightarrow_{ce}{}^* (a1,\ a2)$ **using** *a-prod cptn-elim-cases*(*2*)
proof −
have *f1*: (Γ, (*P*, *Normal s*) # (*a1*, *a2*) # *as*) $\in$ *cptn*
using *a-prod p1* **by** *fastforce*
have *last* [(*a1*, *a2*)] = (*a1*, *a2*)
by *auto*

thus *?thesis*
using *f1* **by** (*metis* (*no-types*) *cptn-dest1 cptn-stepconf-rtrancl last-ConsR not-Cons-self2*)
qed
then have $\Gamma\vdash_c$ (*fst a*, *snd a*) $\rightarrow_{ce}{}^*$ (*la1*,*la2*)
proof −
from *p1* **have** (Γ,(*a* # *as*)) ∈ *cptn* **using** *a-prod cptn-dest* **by** *blast*
thus *?thesis* **by** (*metis cptn-stepconf-rtrancl last-ConsR last-prod list.distinct*(*1*) *prod.collapse*)
qed
then obtain *bb* **where** *Normal bb* = *la2* **using** *last-prod p2* **by** *auto*
thus *?thesis* **by** (*metis* (*no-types*) ‹$\Gamma\vdash_c$ (*fst a*, *snd a*) $\rightarrow_{ce}{}^*$ (*la1*, *la2*)› *steps-ce-not-Normal*)
qed

lemma *lift-P1*:
assumes *map-cptn*:(Γ, *map* (*lift Q*) ((*P*, *s*) # *xs*)) ∈ *cptn* **and**
P-ends:*fst* (*last* ((*P*, *s*) # *xs*)) = *Skip*
shows (Γ, *map* (*lift Q*) ((*P*, *s*) # *xs*) @ [(*Q*, *snd* (*last* ((*P*, *s*) # *xs*)))]) ∈ *cptn*
using *map-cptn P-ends*
proof (*induct xs arbitrary*: *P s*)
case *Nil*
have *P0-skips*: *P*=*Skip* **using** *Nil.prems*(*2*) **by** *auto*
have (Γ,[(*Seq Skip Q*, *s*), (*Q*, *s*)]) ∈ *cptn*
by (*simp add*: *cptn.CptnComp SeqSkipc cptn.CptnOne*)
then show *?case* **using** *P0-skips* **by** (*simp add*: *lift-def*)
next
case (*Cons a xs*)
have (Γ, *map* (*lift Q*) ((*P*, *s*) # *a* # *xs*)) ∈ *cptn*
using *Cons.prems*(*1*) **by** *blast*
have *fst* (*last* (*a* # *xs*)) = *Skip* **using** *Cons.prems*(*2*) **by** *auto*
also have *seq-PQ*:(Γ,(*Seq P Q*,*s*) # (*map* (*lift Q*) (*a*#*xs*))) ∈ *cptn*
by (*metis Cons.prems*(*1*) *Cons-lift*)
then have (Γ,(*map* (*lift Q*) (*a*#*xs*))) ∈ *cptn*
proof −
assume *a1*:(Γ, (*Seq P Q*, *s*) # *map* (*lift Q*) (*a* # *xs*)) ∈ *cptn*
then obtain *a1 a2 xs1* **where** *a2*: *map* (*lift Q*) (*a*#*xs*) = ((*a1*,*a2*)#*xs1*) **by** *fastforce*
thus *?thesis* **using** *cptn-dest* **using** *seq-PQ* **by** *auto*
qed
then have (Γ, *map* (*lift Q*) (*a*#*xs*) @ [(*Q*, *snd* (*last* ((*a*#*xs*))))]) ∈ *cptn*
by (*metis Cons.hyps*(*1*) *calculation prod.collapse*)
then have *t1*:(Γ, (*Seq* (*fst a*) *Q*, (*snd a*))#*map* (*lift Q*) *xs* @ [(*Q*, *snd* (*last* ((*P*, *s*)#(*a*#*xs*))))]) ∈ *cptn*
by (*simp add*: *Cons-lift-append*)
then have (Γ,(*Seq P Q*,*s*) # (*Seq* (*fst a*) *Q*, (*snd a*))#*map* (*lift Q*) *xs*)∈ *cptn*
using *seq-PQ* **by** (*simp add*: *Cons-lift*)
then have *t2*: (Γ,(*Seq P Q*,*s*) # [(*Seq* (*fst a*) *Q*, (*snd a*))]) ∈ *cptn*
using *cptn-dest1* **by** *blast*

then have((*Seq P Q*,*s*) # [(*Seq* (*fst a*) *Q*, (*snd a*))])!*length* [(*Seq* (*fst a*) *Q*, (*snd a*))] = (*Seq* (*fst a*) *Q*, (*snd a*))
by *auto*
then have (Γ,(*Seq P Q*,*s*) # [(*Seq* (*fst a*) *Q*, (*snd a*))]@*map* (*lift Q*) *xs* @ [(*Q*, *snd* (*last* ((*P*, *s*)#(*a*#*xs*))))])∈ *cptn*
using *cptn-append-is-cptn t1 t2* **by** *blast*
then have (Γ, *map* (*lift Q*) ((*P*,*s*)#(*fst a*, *snd a*)#*xs*) @[(*Q*, *snd* (*last* ((*P*, *s*)#(*a*#*xs*))))])∈*cptn*
using *Cons-lift-append append-Cons append-Nil* **by** *metis*
thus *?case* **by** *auto*
qed

lemma *lift-catch-P1*:
assumes *map-cptn*:(Γ, *map* (*lift-catch Q*) ((*P*, *Normal s*) # *xs*)) ∈ *cptn* **and**
P-ends:*fst* (*last* ((*P*, *Normal s*) # *xs*)) = *Throw* **and**
P-ends-normal:∃ *p*. *snd*(*last* ((*P*, *Normal s*) # *xs*)) = *Normal p*
shows (Γ, *map* (*lift-catch Q*) ((*P*, *Normal s*) # *xs*) @ [(*Q*, *snd* (*last* ((*P*, *Normal s*) # *xs*)))]) ∈ *cptn*
using *map-cptn P-ends P-ends-normal*
proof (*induct xs arbitrary*: *P s*)
case *Nil*
have *P0-skips*: *P*=*Throw* **using** *Nil.prems*(*2*) **by** *auto*
have (Γ,[(*Catch Throw Q*, *Normal s*), (*Q*, *Normal s*)]) ∈ *cptn*
by (*simp add*: *cptn.CptnComp CatchThrowc cptn.CptnOne*)
then show *?case* **using** *P0-skips* **by** (*simp add*: *lift-catch-def*)
next
case (*Cons a xs*)
have *s1*:(Γ, *map* (*lift-catch Q*) ((*P*, *Normal s*) # *a* # *xs*)) ∈ *cptn*
using *Cons.prems*(*1*) **by** *blast*
have *s2*:*fst* (*last* (*a* # *xs*)) = *Throw* **using** *Cons.prems*(*2*) **by** *auto*
then obtain *p* **where** *s3*:*snd*(*last* (*a* #*xs*)) = *Normal p* **using** *Cons.prems*(*3*) **by** *auto*
also have *seq-PQ*:(Γ,(*Catch P Q*,*Normal s*) # (*map* (*lift-catch Q*) (*a*#*xs*))) ∈ *cptn*
by (*metis Cons.prems*(*1*) *Cons-lift-catch*) **thm** *Cons.hyps*
then have *axs-in-cptn*:(Γ,(*map* (*lift-catch Q*) (*a*#*xs*))) ∈ *cptn*
proof −
assume *a1*:(Γ, (*Catch P Q*, *Normal s*) # *map* (*lift-catch Q*) (*a* # *xs*)) ∈ *cptn*
then obtain *a1 a2 xs1* **where** *a2*: *map* (*lift-catch Q*) (*a*#*xs*) = ((*a1*,*a2*)#*xs1*) **by** *fastforce*
thus *?thesis* **using** *cptn-dest* **using** *seq-PQ* **by** *auto*
qed
then have (Γ, *map* (*lift-catch Q*) (*a*#*xs*) @ [(*Q*, *snd* (*last* ((*a*#*xs*))))]) ∈ *cptn*
proof (*cases xs*=[])
case *True* **thus** *?thesis* **using** *s2 s3 axs-in-cptn* **by** (*metis Cons.hyps eq-snd-iff last-ConsL*)
next

case *False*
from *seq-PQ* **have** *seq*:(Γ,(*Catch P Q*,*Normal s*) # (*Catch* (*fst a*) *Q*,*snd a*)#*map* (*lift-catch Q*) *xs*)∈ *cptn*
by (*simp add*: *Cons-lift-catch*)
obtain *cf sf* **where** *last-map-axs*:(*cf*,*sf*)=*last* (*map* (*lift-catch Q*) (*a*#*xs*)) **using** *prod.collapse* **by** *blast*
have ∀ *p ps*. (*ps*=[] ∧ *last* [*p*] = *p*) ∨ (*ps*≠[] ∧ *last* (*p*#*ps*) = *last ps*) **by** *simp*
then have *tranclos*:$\Gamma\vdash_c$ (*Catch P Q*,*Normal s*) $\rightarrow_{ce}{}^*$ (*Catch* (*fst a*) *Q*,*snd a*) **using** *Cons-lift-catch*
by (*metis* (*no-types*) *cptn-dest1 cptn-stepc-rtrancl not-Cons-self2 seq*)
have *tranclos-a*:$\Gamma\vdash_c$ (*Catch* (*fst a*) *Q*,*snd a*) $\rightarrow_{ce}{}^*$ (*cf*,*sf*)
by (*metis Cons-lift-catch axs-in-cptn cptn-stepc-rtrancl last-map-axs prod.collapse*)
have *snd-last*:*snd* (*last* (*map* (*lift-catch Q*) (*a*#*xs*))) = *snd* (*last* (*a* #*xs*))
proof −
have *eqslist*:*snd*(((*map* (*lift-catch Q*) (*a*#*xs*)))!(*length* (*map* (*lift-catch Q*) *xs*)))= *snd*((*a*#*xs*)!(*length xs*))
using *last-snd-catch* **by** *fastforce*
have (*lift-catch Q a*)#(*map* (*lift-catch Q*) *xs*) = (*map* (*lift-catch Q*) (*a*#*xs*)) **by** *auto*
then have (*map* (*lift-catch Q*) (*a*#*xs*))!(*length* (*map* (*lift-catch Q*) *xs*)) = *last* (*map* (*lift-catch Q*) (*a*#*xs*))
using *last-length* [*of* (*lift-catch Q a*) (*map* (*lift-catch Q*) *xs*)] **by** *auto*
thus *?thesis* **using** *eqslist* **by** (*simp add*:*last-length*)
qed
then obtain *p1* **where** (*snd a*) = *Normal p1*
by (*metis tranclos-a last-map-axs s3 snd-conv step-ce-normal-to-normal tranclos*)
moreover obtain *a1 a2* **where** *aeq*:*a* = (*a1*,*a2*) **by** *fastforce*
moreover have *fst* (*last* ((*a1*,*a2*) # *xs*)) = *Throw* **using** *s2 False* **by** *auto*
moreover have (Γ, *map* (*lift-catch Q*) ((*a1*,*a2*) # *xs*)) ∈ *cptn* **using** *aeq axs-in-cptn False* **by** *auto*
moreover have ∃ *p*. *snd* (*last* ((*a1*,*a2*) # *xs*)) = *Normal p* **using** *s3 aeq* **by** *auto*
moreover have *a2* = *Normal p1* **using** *aeq calculation*(*1*) **by** *auto*
ultimately have (Γ, *map* (*lift-catch Q*) ((*a1*,*a2*) # *xs*) @
[(*Q*, *snd* (*last* ((*a1*,*a2*) # *xs*)))])∈ *cptn*
using *Cons.hyps aeq* **by** *blast*
thus *?thesis* **using** *aeq* **by** *force*
qed
then have *t1*:(Γ, (*Catch* (*fst a*) *Q*, (*snd a*))#*map* (*lift-catch Q*) *xs* @ [(*Q*, *snd* (*last* ((*P*, *Normal s*)#(*a*#*xs*))))]) ∈ *cptn*
by (*simp add*: *Cons-lift-catch-append*)
then have (Γ,(*Catch P Q*,*Normal s*) # (*Catch* (*fst a*) *Q*, (*snd a*))#*map* (*lift-catch Q*) *xs*)∈ *cptn*
using *seq-PQ* **by** (*simp add*: *Cons-lift-catch*)
then have *t2*: (Γ,(*Catch P Q*,*Normal s*) # [(*Catch* (*fst a*) *Q*, (*snd a*))]) ∈ *cptn*
using *cptn-dest1* **by** *blast*

then have((Catch P Q,Normal s) # [(Catch (fst a) Q, (snd a))])!length [(Catch (fst a) Q, (snd a))] = (Catch (fst a) Q, (snd a))
by auto
then have (Γ,(Catch P Q,Normal s) # [(Catch (fst a) Q, (snd a))]@map (lift-catch Q) xs @ [(Q, snd (last ((P, Normal s)#(a#xs))))])∈ cptn
using cptn-append-is-cptn t1 t2 **by** blast
then have (Γ, map (lift-catch Q) ((P,Normal s)#(fst a, snd a)#xs) @[(Q, snd (last ((P,Normal s)#(a#xs))))])∈cptn
using Cons-lift-catch-append append-Cons append-Nil **by** metis
thus ?case **by** auto
qed

lemma seq2:
assumes
p1:(Γ, (P0, s) # xs) ∈ cptn-mod **and**
p2:(Γ, (P0, s) # xs) ∈ cptn **and**
p3:fst (last ((P0, s) # xs)) = Skip **and**
p4:(Γ, (P1, snd (last ((P0, s) # xs))) # ys) ∈ cptn-mod **and**
p5:(Γ, (P1, snd (last ((P0, s) # xs))) # ys) ∈ cptn **and**
p6:zs = map (lift P1) xs @ (P1, snd (last ((P0, s) # xs))) # ys
shows (Γ, (Seq P0 P1, s) # zs) ∈ cptn
using p1 p2 p3 p4 p5 p6
proof −
have last-skip:fst (last ((P0, s) # xs)) = Skip **using** p3 **by** blast
have (Γ, (map (lift P1) ((P0, s) # xs))@(P1, snd (last ((P0, s) # xs))) # ys) ∈ cptn
proof −
have (Γ,map (lift P1) ((P0, s) #xs)) ∈ cptn
using p2 lift-is-cptn **by** blast
then have (Γ,map (lift P1) ((P0, s) #xs)@[(P1, snd (last ((P0, s) # xs)))]) ∈ cptn
using last-skip lift-P1 **by** blast
then have (Γ,(Seq P0 P1, s) # map (lift P1) xs@[(P1, snd (last ((P0, s) # xs)))]) ∈ cptn
by (simp add: Cons-lift-append)
moreover have last ((Seq P0 P1, s) # map (lift P1) xs @[(P1, snd (last ((P0, s) # xs)))]) = (P1, snd (last ((P0, s) # xs)))
by auto
moreover have last ((Seq P0 P1, s) # map (lift P1) xs @[(P1, snd (last ((P0, s) # xs)))]) =
((Seq P0 P1, s) # map (lift P1) xs @[(P1, snd (last ((P0, s) # xs)))])!length (map (lift P1) xs @[(P1, snd (last ((P0, s) # xs)))])
by (metis last-length)
ultimately have (Γ, (Seq P0 P1, s) # map (lift P1) xs @ (P1, snd (last ((P0, s) # xs))) # ys)∈ cptn
using cptn-append-is-cptn p5 **by** fastforce
thus ?thesis **by** (simp add: Cons-lift-append)
qed
thus ?thesis

by (*simp add*: *Cons-lift-append p6*)
qed

lemma *seq3*:
assumes
p1:(Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod* **and**
p2:(Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn* **and**
p3:*fst* (*last* ((*P0*, *Normal s*) # *xs*)) = *Throw* **and**
p4:*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s′* **and**
p5:(Γ,(*Throw*,*Normal s′*)#*ys*) ∈ *cptn-mod* **and**
p6:(Γ,(*Throw*,*Normal s′*)#*ys*) ∈ *cptn* **and**
p7:*zs* = *map* (*lift P1*) *xs* @((*Throw*,*Normal s′*)#*ys*)
shows (Γ, (*Seq P0 P1*, *Normal s*) # *zs*) ∈ *cptn*
using *p1 p2 p3 p4 p5 p6 p7*
proof (*induct xs arbitrary*: *zs P0 s*)
case *Nil* **thus** *?case* **using** *SeqThrowc cptn.simps* **by** *fastforce*
next
case (*Cons a as*)
then obtain *sa* **where** *snd a* = *Normal sa* **by** (*meson Normal-Normal*)
obtain *a1 a2* **where** *a-prod*:*a*=(*a1*,*a2*) **by** *fastforce*
obtain *la1 la2* **where** *last-prod*:*last* (*a*#*as*) = (*la1*,*la2*) **by** *fastforce*
then have *lasst-aas-last*: *last* (*a*#*as*) = (*last* ((*P0*, *Normal s*) # *a* # *as*)) **by** *auto*
then have *la1* = *Throw* **using** *Cons.prems*(*3*) *last-prod* **by** *force*
have *la2* = *Normal s′* **using** *Cons.prems*(*4*) *last-prod lasst-aas-last* **by** *force*
have *f1*: (Γ, (*a1*, *a2*) # *as*) ∈ *cptn*
using *Cons.prems*(*2*) *a-prod cptn-dest* **by** *blast*
have *f2*: *Normal sa* = *a2*
using ‹*snd a* = *Normal sa*› *a-prod* **by** *force*
have (Γ, *a* # *as*) ∈ *cptn-mod*
using *f1 a-prod cptn-onlyif-cptn-mod* **by** *blast*
then have *hyp*:(Γ, (*Seq a1 P1*, *Normal sa*) #
map (*lift P1*) *as* @ ((*Throw*,*Normal s′*)#*ys*)) ∈ *cptn*
using *Cons.hyps Cons.prems*(*3*) *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*6*) *a-prod f1 f2* **by** *fastforce*
thus *?case*
proof −
have (*Seq a1 P1*, *a2*) # *map* (*lift P1*) *as* @((*Throw*,*Normal s′*)#*ys*) = *zs*
by (*simp add*: *Cons.prems*(*7*) *Cons-lift-append a-prod*)
thus *?thesis*
by (*metis* (*no-types*, *lifting*) *Cons.prems*(*2*) *Seqc a-prod cptn.CptnComp cptn.CptnEnv Env cptn-elim-cases*(*2*) *f2 hyp*)
qed
qed

lemma *cptn-if-cptn-mod*:
assumes *cptn-mod-asm*:(Γ,*c*) ∈ *cptn-mod*
shows (Γ,*c*) ∈ *cptn*
using *cptn-mod-asm*

proof (*induct*)
 case (*CptnModOne*) **thus** *?case* **using** *cptn.CptnOne* **by** *blast*
next
 case *CptnModSkip* **thus** *?case* **by** (*simp add*: *cptn.CptnComp*)
next
 case *CptnModThrow* **thus** *?case* **by** (*simp add*: *cptn.CptnComp*)
next
 case *CptnModCondT* **thus** *?case* **by** (*simp add*: *CondTruec cptn.CptnComp*)
next
 case *CptnModCondF* **thus** *?case* **by** (*simp add*: *CondFalsec cptn.CptnComp*)
next
 case (*CptnModSeq1* Γ *P0 s xs zs P1*)
 have (Γ, *map* (*lift P1*) ((*P0*, *s*) # *xs*)) ∈ *cptn*
  using *CptnModSeq1.hyps*(*2*) *lift-is-cptn* **by** *blast*
 thus *?case* **by** (*simp add*: *Cons-lift CptnModSeq1.hyps*(*3*))
next
 case (*CptnModSeq2* Γ *P0 s xs P1 ys zs*)
 thus *?case* **by** (*simp add*:*seq2*)
next
 case (*CptnModSeq3* Γ *P0 s xs s′ zs P1*)
 thus *?case* **by** (*simp add*: *seq3*)
next
 case (*CptnModWhile1* Γ *P s xs b zs*) **thus** *?case* **by** (*metis Cons-lift WhileTruec cptn.CptnComp lift-is-cptn*)
next
 case (*CptnModWhile2* Γ *P s xs b zs ys*)
 then have (Γ, (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn*
  by (*simp add*:*seq2*)
 then have Γ$\vdash_c$(*While b P*,*Normal s*) → (*Seq P* (*While b P*),*Normal s*)
  by (*simp add*: *CptnModWhile2.hyps*(*4*) *WhileTruec*)
 thus *?case*
 by (*simp add*: ‹(Γ, (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn*› *cptn.CptnComp*)

next
 case (*CptnModWhile3* Γ *P s xs b s′ ys zs*)
 then have (Γ,(*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn*
  by (*simp add*: *seq3*)
 then have Γ$\vdash_c$(*While b P*,*Normal s*) → (*Seq P* (*While b P*),*Normal s*) **by** (*simp add*: *CptnModWhile3.hyps*(*4*) *WhileTruec*)
 thus *?case* **by** (*simp add*: ‹(Γ, (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn*› *cptn.CptnComp*)
next
 case (*CptnModCall* Γ *bdy s ys p*) **thus** *?case* **by** (*simp add*: *Callc cptn.CptnComp*)

next
 case (*CptnModDynCom* Γ *c s ys*) **thus** *?case* **by** (*simp add*: *DynComc cptn.CptnComp*)
next
 case (*CptnModGuard* Γ *c s ys g f*) **thus** *?case* **by** (*simp add*: *Guardc cptn.CptnComp*)
next

```
  case (CptnModCatch1 Γ P0 s xs zs P1)
  have (Γ, map (lift-catch P1) ((P0, s) # xs)) ∈ cptn
    using CptnModCatch1.hyps(2) lift-catch-is-cptn by blast
  thus ?case by (simp add: Cons-lift-catch CptnModCatch1.hyps(3))
next
  case (CptnModCatch2 Γ P0 s xs ys zs P1)
  thus ?case
  proof (induct xs arbitrary: zs P0 s)
    case Nil thus ?case using CatchSkipc cptn.simps by fastforce
  next
    case (Cons a as)
    then obtain sa where snd a = sa by auto
    then obtain a1 a2 where a-prod:a=(a1,a2) and sa-a2: a2 =sa
          by fastforce
    obtain la1 la2 where last-prod:last (a#as) = (la1,la2) by fastforce
    then have lasst-aas-last: last (a#as) = (last ((P0, s) # a # as)) by auto
    then have la1 = Skip using Cons.prems(3) last-prod by force
    have f1: (Γ, (a1, a2) # as) ∈ cptn
      using Cons.prems(2) a-prod cptn-dest by blast
    have (Γ, a # as) ∈ cptn-mod
      using f1 a-prod cptn-onlyif-cptn-mod by blast
    then have hyp:(Γ, (Catch a1 P1, a2) #
            map (lift-catch P1) as @ ((Skip, la2)#ys)) ∈ cptn
        using Cons.hyps Cons.prems a-prod f1 last-prod by fastforce
    thus ?case
    proof -
      have f1:(Catch a1 P1, a2) # map (lift-catch P1) as @ ((Skip, la2)#ys) = zs
       using Cons.prems(4) Cons-lift-catch-append a-prod last-prod by (simp add:
Cons.prems(6))
      have (Γ, map (lift-catch P1) ((P0, s) # a # as)) ∈ cptn
       using Cons.prems(2) lift-catch-is-cptn by blast
      hence (Γ, (LanguageCon.com.Catch P0 P1, s) # (LanguageCon.com.Catch
a1 P1, a2) # map (lift-catch P1) as) ∈ cptn
        by (metis (no-types) Cons-lift-catch a-prod)
    hence (Γ, (LanguageCon.com.Catch P0 P1, s) # zs) ∈ cptn ∨ (Γ, (LanguageCon.com.Catch
P0 P1, s) # (LanguageCon.com.Catch a1 P1, a2) # map (lift-catch P1) as) ∈
cptn ∧ (¬ Γ⊢c (LanguageCon.com.Catch P0 P1, s) →e (LanguageCon.com.Catch
P0 P1, a2) ∨ (Γ, (LanguageCon.com.Catch P0 P1, a2) # map (lift-catch P1) as)
∉ cptn ∨ LanguageCon.com.Catch a1 P1 ≠ LanguageCon.com.Catch P0 P1)
        using f1 cptn.CptnEnv hyp by blast
      thus ?thesis
       by (metis (no-types) f1 cptn.CptnComp cptn-elim-cases(2) hyp)
    qed
  qed
next
  case (CptnModCatch3 Γ P0 s xs s' P1 ys zs)
  thus ?case
  proof (induct xs arbitrary: zs P0 s)
    case Nil thus ?case using CatchThrowc cptn.simps by fastforce
```

next
case (*Cons a as*)
then obtain *sa* **where** *snd a* = *Normal sa* **by** (*meson Normal-Normal*)
obtain *a1 a2* **where** *a-prod*:*a*=(*a1*,*a2*) **by** *fastforce*
obtain *la1 la2* **where** *last-prod*:*last* (*a#as*) = (*la1*,*la2*) **by** *fastforce*
then have *lasst-aas-last*: *last* (*a#as*) = (*last* ((*P0*, *Normal s*) # *a* # *as*)) **by** *auto*
then have *la1* = *Throw* **using** *Cons.prems*(*3*) *last-prod* **by** *force*
have *la2* = *Normal s'* **using** *Cons.prems*(*4*) *last-prod lasst-aas-last* **by** *force*
have *f1*: (Γ, (*a1*, *a2*) # *as*) ∈ *cptn*
using *Cons.prems*(*2*) *a-prod cptn-dest* **by** *blast*
have *f2*: *Normal sa* = *a2*
using ‹*snd a* = *Normal sa*› *a-prod* **by** *force*
have (Γ, *a* # *as*) ∈ *cptn-mod*
using *f1 a-prod cptn-onlyif-cptn-mod* **by** *blast*
then have *hyp*:(Γ, (*Catch a1 P1*, *Normal sa*) #
map (*lift-catch P1*) *as* @ (*P1*, *snd* (*last* ((*a1*, *Normal sa*) # *as*))) # *ys*) ∈ *cptn*
using *Cons.hyps Cons.prems a-prod f1 f2* **by** *auto*
thus *?case*
proof −
have $\Gamma\vdash_c$ (*P0*, *Normal s*) $\rightarrow_e$ (*P0*, *a2*)
by (*fastforce intro*: *step-e.intros*)
then have *transit*:$\Gamma\vdash_c$(*P0*,*Normal s*) $\rightarrow_{ce}$ (*a1*,*Normal sa*)
by (*metis* (*no-types*) *Cons.prems*(*2*) *a-prod c-step cptn-elim-cases*(*2*) *e-step f2*)
then have *transit-catch*:$\Gamma\vdash_c$(*Catch P0 P1*,*Normal s*) $\rightarrow_{ce}$ (*Catch a1 P1*,*Normal sa*)
by (*metis* (*no-types*) *Catchc c-step e-step env-c-c' step-ce-elim-cases step-e.intros*(*1*))
have (*Catch a1 P1*, *a2*) # *map* (*lift-catch P1*) *as* @ (*P1*, *la2*) # *ys* = *zs*
using *Cons.prems Cons-lift-catch-append a-prod last-prod* **by** *auto*
have *a*=(*a1*, *Normal sa*) **using** *a-prod f2* **by** *auto*
have *snd* (*last* ((*a1*, *Normal sa*) # *as*)) = *Normal s'*
using ‹*a* = (*a1*, *Normal sa*)› ‹*snd* (*last* ((*P0*, *Normal s*) # *a* # *as*)) = *Normal s'*› *lasst-aas-last* **by** *fastforce*
hence *f1*: *snd* (*last* ((*a1*, *Normal sa*) # *as*)) = *la2*
using ‹*la2* = *Normal s'*› **by** *blast*
have $\Gamma\vdash_c$ (*LanguageCon.com.Catch P0 P1*, *Normal s*) $\rightarrow_{ce}$ (*LanguageCon.com.Catch a1 P1*, *a2*)
using *f2 transit-catch* **by** *blast*
thus *?thesis*
using *f1* ‹(*LanguageCon.com.Catch a1 P1*, *a2*) # *map* (*lift-catch P1*) *as* @ (*P1*, *la2*) # *ys* = *zs*›
cptn.CptnComp cptn.CptnEnv f2 hyp not-eq-not-env step-ce-not-step-e-step-c

by *metis*
qed
qed

next
  case (*CptnModEnv*) **thus** *?case* **by** (*simp add*: *cptn.CptnEnv*)
qed

lemma *cptn-eq-cptn-mod*:
shows $(x \in cptn\text{-}mod) = (x \in cptn)$
by (*cases x*, *auto simp add*: *cptn-if-cptn-mod cptn-onlyif-cptn-mod*)

lemma *cptn-eq-cptn-mod-set*:
shows $cptn\text{-}mod = cptn$
by (*auto simp add*: *cptn-if-cptn-mod cptn-onlyif-cptn-mod*)

8.8 Computational modular semantic for nested calls

inductive-set *cptn-mod-nest-call* :: $(nat \times ('s,'p,'f,'e)\ confs)\ set$
where
  CptnModNestOne: $(n,\Gamma,[(P,\ s)]) \in cptn\text{-}mod\text{-}nest\text{-}call$
| *CptnModNestEnv*: $[\![\Gamma\vdash_c(P,s) \rightarrow_e (P,t);(n,\Gamma,(P,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call]\!] \Longrightarrow$
$(n,\Gamma,(P,\ s)\#(P,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call$
| *CptnModNestSkip*: $[\![\Gamma\vdash_c(P,s) \rightarrow (Skip,t);\ redex\ P = P;$
$\forall f.\ ((\exists sn.\ s = Normal\ sn) \wedge (\Gamma\ f) = Some\ Skip \longrightarrow P \neq Call\ f\ );$
$(n,\Gamma,(Skip,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ ]\!] \Longrightarrow$
$(n,\Gamma,(P,s)\#(Skip,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call$

| *CptnModNestThrow*: $[\![\Gamma\vdash_c(P,s) \rightarrow (Throw,t);\ redex\ P = P;$
$\forall f.\ ((\exists sn.\ s = Normal\ sn) \wedge (\Gamma\ f) = Some\ Throw \longrightarrow P \neq Call\ f\ );$
$(n,\Gamma,(Throw,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ ]\!] \Longrightarrow$
$(n,\Gamma,(P,s)\#(Throw,\ t)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call$

| *CptnModNestCondT*: $[\![(n,\Gamma,(P0,\ Normal\ s)\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call;\ s \in b\ ]\!] \Longrightarrow$
$(n,\Gamma,((Cond\ b\ P0\ P1),\ Normal\ s)\#(P0,\ Normal\ s)\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call$

| *CptnModNestCondF*: $[\![(n,\Gamma,(P1,\ Normal\ s)\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call;\ s \notin b\ ]\!] \Longrightarrow$
$(n,\Gamma,((Cond\ b\ P0\ P1),\ Normal\ s)\#(P1,\ Normal\ s)\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call$

| *CptnModNestSeq1*:
  $[\![(n,\Gamma,(P0,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call;\ zs=map\ (lift\ P1)\ xs\ ]\!] \Longrightarrow$
  $(n,\Gamma,((Seq\ P0\ P1),\ s)\#zs) \in cptn\text{-}mod\text{-}nest\text{-}call$

| *CptnModNestSeq2*:
  $[\![(n,\Gamma,\ (P0,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call;\ fst(last\ ((P0,\ s)\#xs)) = Skip;$
  $(n,\Gamma,(P1,\ snd(last\ ((P0,\ s)\#xs)))\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call;$

zs=(*map* (*lift P1*) *xs*)@((*P1*, *snd*(*last* ((*P0*, *s*)#*xs*)))#*ys*) $]\!]$ $\Longrightarrow$
(*n*,Γ,((*Seq P0 P1*), *s*)#*zs*) $\in$ *cptn-mod-nest-call*

| *CptnModNestSeq3*:
$[\![$(*n*,Γ, (*P0*, *Normal s*)#*xs*) $\in$ *cptn-mod-nest-call*;
fst(*last* ((*P0*, *Normal s*)#*xs*)) = *Throw*;
snd(*last* ((*P0*, *Normal s*)#*xs*)) = *Normal s'*;
(*n*,Γ,(*Throw*,*Normal s'*)#*ys*) $\in$ *cptn-mod-nest-call*;
zs=(*map* (*lift P1*) *xs*)@((*Throw*,*Normal s'*)#*ys*) $]\!]$ $\Longrightarrow$
(*n*,Γ,((*Seq P0 P1*), *Normal s*)#*zs*) $\in$ *cptn-mod-nest-call*

| *CptnModNestWhile1*:
$[\![$(*n*,Γ, (*P*, *Normal s*)#*xs*) $\in$ *cptn-mod-nest-call*; *s* $\in$ *b*;
zs=*map* (*lift* (*While b P*)) *xs* $]\!]$ $\Longrightarrow$
(*n*,Γ, ((*While b P*), *Normal s*)#
((*Seq P* (*While b P*)),*Normal s*)#*zs*) $\in$ *cptn-mod-nest-call*

| *CptnModNestWhile2*:
$[\![$ (*n*,Γ, (*P*, *Normal s*)#*xs*) $\in$ *cptn-mod-nest-call*;
fst(*last* ((*P*, *Normal s*)#*xs*))=*Skip*; *s* $\in$ *b*;
zs=(*map* (*lift* (*While b P*)) *xs*)@
(*While b P*, *snd*(*last* ((*P*, *Normal s*)#*xs*)))#*ys*;
(*n*,Γ,(*While b P*, *snd*(*last* ((*P*, *Normal s*)#*xs*)))#*ys*) $\in$
cptn-mod-nest-call$]\!]$ $\Longrightarrow$
(*n*,Γ,(*While b P*, *Normal s*)#
(*Seq P* (*While b P*), *Normal s*)#*zs*) $\in$ *cptn-mod-nest-call*

| *CptnModNestWhile3*:
$[\![$ (*n*,Γ, (*P*, *Normal s*)#*xs*) $\in$ *cptn-mod-nest-call*;
fst(*last* ((*P*, *Normal s*)#*xs*))=*Throw*; *s* $\in$ *b*;
snd(*last* ((*P*, *Normal s*)#*xs*)) = *Normal s'*;
(*n*,Γ,(*Throw*,*Normal s'*)#*ys*) $\in$ *cptn-mod-nest-call*;
zs=(*map* (*lift* (*While b P*)) *xs*)@((*Throw*,*Normal s'*)#*ys*)$]\!]$ $\Longrightarrow$
(*n*,Γ,(*While b P*, *Normal s*)#
(*Seq P* (*While b P*), *Normal s*)#*zs*) $\in$ *cptn-mod-nest-call*

| *CptnModNestCall*: $[\![$(*n*,Γ,(*bdy*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call*;Γ *p* = *Some bdy*; *bdy*$\neq$*Call p* $]\!]$ $\Longrightarrow$
(*Suc n*, Γ,((*Call p*), *Normal s*)#(*bdy*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call*

| *CptnModNestDynCom*: $[\![$(*n*,Γ,(*c s*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call* $]\!]$ $\Longrightarrow$
(*n*,Γ,(*DynCom c*, *Normal s*)#(*c s*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call*

| *CptnModNestGuard*: $[\![$(*n*,Γ,(*c*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call*; *s* $\in$ *g* $]\!]$ $\Longrightarrow$
(*n*,Γ,(*Guard f g c*, *Normal s*)#(*c*, *Normal s*)#*ys*) $\in$ *cptn-mod-nest-call*

| *CptnModNestCatch1*: $[\![$(*n*,Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call*; *zs*=*map* (*lift-catch P1*) *xs* $]\!]$

$\Longrightarrow (n,\Gamma,((Catch\ P0\ P1),\ s)\#zs) \in$ *cptn-mod-nest-call*

| *CptnModNestCatch2*:
$[\![(n,\Gamma,\ (P0,\ s)\#xs) \in$ *cptn-mod-nest-call*; $fst(last\ ((P0,\ s)\#xs)) = Skip$;
$(n,\Gamma,(Skip,snd(last\ ((P0,\ s)\#xs)))\#ys) \in$ *cptn-mod-nest-call*;
$zs=(map\ (lift\text{-}catch\ P1)\ xs)@((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)\ ]\!] \Longrightarrow$
$(n,\Gamma,((Catch\ P0\ P1),\ s)\#zs) \in$ *cptn-mod-nest-call*

| *CptnModNestCatch3*:
$[\![(n,\Gamma,\ (P0,\ Normal\ s)\#xs) \in$ *cptn-mod-nest-call*; $fst(last\ ((P0,\ Normal\ s)\#xs))$
$= Throw$;
$snd(last\ ((P0,\ Normal\ s)\#xs)) = Normal\ s'$;
$(n,\Gamma,(P1,\ snd(last\ ((P0,\ Normal\ s)\#xs)))\#ys) \in$ *cptn-mod-nest-call*;
$zs=(map\ (lift\text{-}catch\ P1)\ xs)@((P1,\ snd(last\ ((P0,\ Normal\ s)\#xs)))\#ys)\ ]\!] \Longrightarrow$
$(n,\Gamma,((Catch\ P0\ P1),\ Normal\ s)\#zs) \in$ *cptn-mod-nest-call*

lemmas *CptnMod-nest-call-induct* = *cptn-mod-nest-call.induct* [*of* - - $[(c,s)]$, *split-format* (*complete*), *case-names*
CptnModOne CptnModEnv CptnModSkip CptnModThrow CptnModCondT CptnModCondF
CptnModSeq1 CptnModSeq2 CptnModSeq3 CptnModSeq4 CptnModWhile1 CptnModWhile2 CptnModWhile3 CptnModCall CptnModDynCom CptnModGuard
CptnModCatch1 CptnModCatch2 CptnModCatch3, *induct set*]

inductive-cases *CptnModNest-elim-cases* [*cases set*]:
$(n,\Gamma,(Skip,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Guard\ f\ g\ c,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Basic\ f\ e,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Spec\ r\ e,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Seq\ c1\ c2,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Cond\ b\ c1\ c2,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Await\ b\ c2\ e,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Call\ p,\ s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(DynCom\ c,s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Throw,s)\#u\#xs) \in$ *cptn-mod-nest-call*
$(n,\Gamma,(Catch\ c1\ c2,s)\#u\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *stepc-elim-cases-Seq-Seq'*:
$\Gamma\vdash_c(Seq\ c1\ c2,s) \rightarrow (Seq\ c1'\ c2',s')$

inductive-cases *stepc-elim-cases-Catch-Catch'*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (Catch\ c1'\ c2',s')$

inductive-cases *CptnModNest-same-elim-cases* [*cases set*]:
$(n,\Gamma,(u,\ s)\#(u,t)\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Stuck* [*cases set*]:
$(n,\Gamma,(P,\ Stuck)\#(Skip,\ s)\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Fault* [*cases set*]:
$(n,\Gamma,(P,\ \mathit{Fault}\ f)\#(\mathit{Skip},\ s)\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Abrupt* [*cases set*]:
$(n,\Gamma,(P,\ \mathit{Abrupt}\ as)\#(\mathit{Skip},\ s)\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Call-Stuck* [*cases set*]:
$(n,\Gamma,(\mathit{Call}\ p,\ s)\#(\mathit{Skip},\ \mathit{Stuck})\#xs) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Call* [*cases set*]:
$(0,\ \Gamma,((\mathit{Call}\ p),\ \mathit{Normal}\ s)\#(bdy,\ \mathit{Normal}\ s)\#ys) \in$ *cptn-mod-nest-call*

inductive-cases *CptnEmpty* [*cases set*]:
$(n,\ \Gamma,[]) \in$ *cptn-mod-nest-call*

inductive-cases *CptnModNest-elim-cases-Call-normal* [*cases set*]:
$(\mathit{Suc}\ n,\ \Gamma,((\mathit{Call}\ p),\ \mathit{Normal}\ s)\#(bdy,\ \mathit{Normal}\ s)\#ys) \in$ *cptn-mod-nest-call*

lemma *cptn-mod-nest-mono1*: $(n,\Gamma,cfs) \in$ *cptn-mod-nest-call* $\Longrightarrow (\mathit{Suc}\ n,\Gamma,cfs)\in$ *cptn-mod-nest-call*
proof (*induct rule*:*cptn-mod-nest-call.induct*)
 case (*CptnModNestOne*) **thus** *?case* **using** *cptn-mod-nest-call.CptnModNestOne* **by** *auto*
next
 case (*CptnModNestEnv*) **thus** *?case* **using** *cptn-mod-nest-call.CptnModNestEnv* **by** *fastforce*
next
 case (*CptnModNestSkip*) **thus** *?case* **using** *cptn-mod-nest-call.CptnModNestSkip* **by** *fastforce*
next
 case (*CptnModNestThrow*) **thus** *?case* **using** *cptn-mod-nest-call.intros*(*4*) **by** *fastforce*
next
 case (*CptnModNestCondT n*) **thus** *?case*
 using *cptn-mod-nest-call.CptnModNestCondT*[*of Suc n*] **by** *fastforce*
next
 case (*CptnModNestCondF n*) **thus** *?case*
 using *cptn-mod-nest-call.CptnModNestCondF*[*of Suc n*] **by** *fastforce*
next
 case (*CptnModNestSeq1 n*) **thus** *?case*
 using *cptn-mod-nest-call.CptnModNestSeq1*[*of Suc n*] **by** *fastforce*
next
 case (*CptnModNestSeq2 n*) **thus** *?case*
 using *cptn-mod-nest-call.CptnModNestSeq2*[*of Suc n*] **by** *fastforce*
next
 case (*CptnModNestSeq3 n*) **thus** *?case*
 using *cptn-mod-nest-call.CptnModNestSeq3*[*of Suc n*] **by** *fastforce*
next
 case (*CptnModNestWhile1 n*) **thus** *?case*

```
    using cptn-mod-nest-call.CptnModNestWhile1[of Suc n] by fastforce
next
  case (CptnModNestWhile2 n) thus ?case
    using cptn-mod-nest-call.CptnModNestWhile2[of Suc n] by fastforce
next
  case (CptnModNestWhile3 n) thus ?case
    using cptn-mod-nest-call.CptnModNestWhile3[of Suc n] by fastforce
next
 case (CptnModNestCall) thus ?case
    using cptn-mod-nest-call.CptnModNestCall by fastforce
next
 case (CptnModNestDynCom) thus ?case
    using cptn-mod-nest-call.CptnModNestDynCom by fastforce
next
 case (CptnModNestGuard n) thus ?case
    using cptn-mod-nest-call.CptnModNestGuard[of Suc n] by fastforce
next
 case (CptnModNestCatch1 n) thus ?case
    using cptn-mod-nest-call.CptnModNestCatch1[of Suc n] by fastforce
next
 case (CptnModNestCatch2 n) thus ?case
    using cptn-mod-nest-call.CptnModNestCatch2[of Suc n] by fastforce
next
 case (CptnModNestCatch3 n) thus ?case
    using cptn-mod-nest-call.CptnModNestCatch3[of Suc n] by fastforce
qed

lemma cptn-mod-nest-mono2:
  (n,Γ,cfs) ∈  cptn-mod-nest-call  ⟹ m>n ⟹
   (m,Γ,cfs)∈ cptn-mod-nest-call
proof (induct m−n arbitrary: m n)
  case 0 thus ?case by auto
next
  case (Suc k)
  have m − Suc n = k
    using Suc.hyps(2) Suc.prems(2) Suc-diff-Suc Suc-inject by presburger
  then show ?case
   using Suc.hyps(1) Suc.prems(1) Suc.prems(2) cptn-mod-nest-mono1 less-Suc-eq
by blast
qed

lemma cptn-mod-nest-mono:
  (n,Γ,cfs) ∈  cptn-mod-nest-call  ⟹ m≥n ⟹
   (m,Γ,cfs)∈ cptn-mod-nest-call
proof (cases n=m)
  assume (n, Γ, cfs) ∈ cptn-mod-nest-call and
         n = m  thus ?thesis by auto
next
  assume (n, Γ, cfs) ∈ cptn-mod-nest-call and
```

$n \leq m$ **and**
$n \neq m$
thus *?thesis* **by** (*auto simp add*: *cptn-mod-nest-mono2*)
qed

8.9 Lemmas on normalization

8.10 Equivalence of comp mod semantics and comp mod nested

definition *catch-cond-nest*
where
catch-cond-nest zs Q xs P s s'' s' Γ *n* $\equiv$ (*zs*=(*map* (*lift-catch Q*) *xs*) $\vee$
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* $\wedge$
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* $\wedge$ *s*=*Normal s''*$\wedge$
($\exists$ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) $\in$ *cptn-mod-nest-call*
$\wedge$
zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*))))
$\vee$
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* $\wedge$
($\exists$ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*) $\in$ *cptn-mod-nest-call* $\wedge$

zs=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*)))))

lemma *div-catch-nest*: **assumes** *cptn-m*:(*n*,Γ,*list*) $\in$ *cptn-mod-nest-call*
shows ($\forall$ *s P Q zs*. *list*=(*Catch P Q*, *s*)#*zs* $\longrightarrow$
($\exists$ *xs s' s''*.
(*n*, Γ,(*P*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* $\wedge$
catch-cond-nest zs Q xs P s s'' s' Γ *n*))

unfolding *catch-cond-nest-def*
using *cptn-m*
proof (*induct rule*: *cptn-mod-nest-call.induct*)
case (*CptnModNestOne* Γ *P s*)
thus *?case* **using** *cptn-mod-nest-call.CptnModNestOne* **by** *blast*
next
case (*CptnModNestSkip* Γ *P s t n xs*)
from *CptnModNestSkip.hyps*
have *step*: $\Gamma \vdash_c$ (*P*, *s*) $\rightarrow$ (*Skip*, *t*) **by** *auto*
from *CptnModNestSkip.hyps*
have *noskip*: ~(*P*=*Skip*) **using** *stepc-elim-cases*(*1*) **by** *blast*
have *no-catch*: $\forall$ *p1 p2*. $\neg$(*P*=*Catch p1 p2*) **using** *CptnModNestSkip.hyps*(*2*)
redex-not-Catch **by** *auto*
from *CptnModNestSkip.hyps*
have *in-cptn-mod*: (*n*,Γ, (*Skip*, *t*) # *xs*) $\in$ *cptn-mod-nest-call* **by** *auto*
then show *?case* **using** *no-catch* **by** *simp*
next
case (*CptnModNestThrow* Γ *P s t n xs*)
from *CptnModNestThrow.hyps*

have *step*: $\Gamma\vdash_c$ $(P,\ s) \rightarrow (Throw,\ t)$ **by** *auto*
from *CptnModNestThrow.hyps*
have *in-cptn-mod*: $(n,\Gamma,\ (Throw,\ t)\ \#\ xs) \in$ *cptn-mod-nest-call* **by** *auto*
have *no-catch*: $\forall\, p1\ p2.\ \neg(P{=}Catch\ p1\ p2)$ **using** *CptnModNestThrow.hyps*(*2*) *redex-not-Catch* **by** *auto*
then show *?case* **by** *auto*
next
case (*CptnModNestCondT* Γ *P0 s ys b P1*)
thus *?case* **using** *CptnModOne* **by** *blast*
next
case (*CptnModNestCondF* Γ *P0 s ys b P1*)
thus *?case* **using** *CptnModOne* **by** *blast*
next
case (*CptnModNestCatch1 sa P Q zs*)
thus *?case* **by** *blast*
next
case (*CptnModNestCatch2 n* Γ *P0 s xs ys zs P1*)
from *CptnModNestCatch2.hyps*(*3*)
have *last*:*fst* $(((P0,\ s)\ \#\ xs)\ !\ length\ xs) = Skip$
by (*simp add*: *last-length*)
have *P0cptn*:$(n,\Gamma,\ (P0,\ s)\ \#\ xs) \in$ *cptn-mod-nest-call* **by** *fact*
then have $zs = map\ (lift\text{-}catch\ P1)\ xs\ @((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)$ **by** (*simp add*:*CptnModNestCatch2.hyps*)
show *?case*
proof −{
fix *sa P Q zsa*
assume *eq*:$(Catch\ P0\ P1,\ s)\ \#\ zs = (Catch\ P\ Q,\ sa)\ \#\ zsa$
then have $P0 = P \wedge P1 = Q \wedge s{=}sa \wedge zs{=}zsa$ **by** *auto*
then have $(P0,\ s) = (P,\ sa)$ **by** *auto*
have $last\ ((P0,\ s)\ \#\ xs) = ((P,\ sa)\ \#\ xs)\ !\ length\ xs$
by (*simp add*: ‹$P0 = P \wedge P1 = Q \wedge s = sa \wedge zs = zsa$› *last-length*)
then have $zs = (map\ (lift\text{-}catch\ Q)\ xs)@((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)$
using ‹$P0 = P \wedge P1 = Q \wedge s = sa \wedge zs = zsa$› ‹$zs = map\ (lift\text{-}catch\ P1)\ xs\ @\ ((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)$›
by *force*
then have $(\exists\, xs\ s'\ s''.\ ((n,\Gamma,(P,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$((zs{=}(map\ (lift\text{-}catch\ Q)\ xs) \vee$
$((fst(((P,\ s)\#xs)!length\ xs){=}Throw \wedge$
$snd(last\ ((P,\ s)\#xs)) = Normal\ s' \wedge\ s{=}Normal\ s''\wedge$
$(\exists\, ys.\ (n,\Gamma,(Q,\ snd(((P,\ s)\#xs)!length\ xs))\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\wedge$
$zs{=}(map\ (lift\text{-}catch\ Q)\ xs)@((Q,\ snd(((P,\ s)\#xs)!length\ xs))\#ys))))$
$\vee$
$(\exists\, ys.\ ((fst(((P,\ s)\#xs)!length\ xs){=}Skip \wedge (n,\Gamma,(Skip,snd(last\ ((P,\ s)\#xs)))\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$zs{=}(map\ (lift\text{-}catch\ Q)\ xs)@((Skip,snd(last\ ((P0,\ s)\#xs)))\#ys)))))))$
using *P0cptn* ‹$P0 = P \wedge P1 = Q \wedge s = sa \wedge zs = zsa$› *last CptnModNestCatch2.hyps*(*4*) **by** *blast*
}

thus *?thesis* **by** *auto*
qed
next
case (*CptnModNestCatch3 n* Γ *P0 s xs s' P1 ys zs*)
from *CptnModNestCatch3.hyps*(*3*)
have *last*:*fst* (((*P0*, *Normal s*) # *xs*) ! *length xs*) = *Throw*
by (*simp add*: *last-length*)
from *CptnModNestCatch3.hyps*(*4*)
have *lastnormal*:*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s'*
by (*simp add*: *last-length*)
have *P0cptn*:(*n*,Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* **by** *fact*
from *CptnModNestCatch3.hyps*(*5*)
have *P1cptn*:(*n*,Γ, (*P1*, *snd* (((*P0*, *Normal s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call*
by (*simp add*: *last-length*)
then have *zs* = *map* (*lift-catch P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *ys*
by (*simp add*:*CptnModNestCatch3.hyps*)
show *?case*
proof −{
fix *sa P Q zsa*
assume *eq*:(*Catch P0 P1*, *Normal s*) # *zs* = (*Catch P Q*, *Normal sa*) # *zsa*
then have *P0* =*P* ∧ *P1* = *Q* ∧ *Normal s*= *Normal sa* ∧ *zs*=*zsa* **by** *auto*
have *last* ((*P0*, *Normal s*) # *xs*) = ((*P*, *Normal sa*) # *xs*) ! *length xs*
by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last-length*)
then have *zsa* = *map* (*lift-catch Q*) *xs* @ (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift-catch P1*) *xs* @ (*P1*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *ys*› **by** *force*
then have (*n*,Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧ (*fst*(((*P*, *Normal s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *Normal s*)#*xs*)) = *Normal s'* ∧
(∃ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *Normal s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *Normal s*)#*xs*)!*length xs*))#*ys*)))
using *lastnormal P1cptn P0cptn* ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last*
by *auto*
}**note** *this* [*of P0 P1 s zs*] **thus** *?thesis* **by** *blast* **qed**
next
case (*CptnModNestEnv* Γ *P s t n xs*)
then have *step*:(*n*, Γ, (*P*, *t*) # *xs*) ∈ *cptn-mod-nest-call* **by** *auto*
have *step-e*: $\Gamma\vdash_c$ (*P*, *s*) $\rightarrow_e$ (*P*, *t*) **using** *CptnModNestEnv* **by** *auto*
show *?case*
proof (*cases P*)
case (*Catch P1 P2*)
then have *eq-P-Catch*:(*P*, *t*) # *xs* = (*LanguageCon.com.Catch P1 P2*, *t*) #

xs **by** *auto*
then obtain *xsa t′ t′′* **where**
p1:(*n*,Γ, (*P1*, *t*) # *xsa*) ∈ *cptn-mod-nest-call* **and**
p2: (*xs* = *map* (*lift-catch P2*) *xsa* ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal t′* ∧
t = *Normal t′′* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift-catch P2*) *xsa* @ (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*.(*n*,Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod-nest-call* ∧

xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*)))
using *CptnModNestEnv*(*3*) **by** *auto*
have *all-step*:(*n*,Γ, (*P1*, *s*)#((*P1*, *t*) # *xsa*)) ∈ *cptn-mod-nest-call*
using *p1 Env Env-n cptn-mod.CptnModEnv env-normal-s step-e*
proof −
have *f1*: *SmallStepCon.redex P* = *SmallStepCon.redex P1*
using *local.Catch* **by** *auto*
obtain *bb* :: (′*b*, ′*c*) *xstate* ⇒ ′*b* **where**
∀ *x2*. (∃ *v5*. *x2* = *Normal v5*) = (*x2* = *Normal* (*bb x2*))
by *moura*
then have *s* = *t* ∨ *s* = *Normal* (*bb s*)
by (*metis* (*no-types*) *env-normal-s step-e*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *Env Env-n cptn-mod-nest-call.CptnModNestEnv p1*)
qed
show *?thesis* **using** *p2*
proof
assume *xs* = *map* (*lift-catch P2*) *xsa*
have (*P*, *t*) # *xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*)
by (*simp add*: ‹*xs* = *map* (*lift-catch P2*) *xsa*› *lift-catch-def local.Catch*)
thus *?thesis* **using** *all-step eq-P-Catch* **by** *fastforce*
next
assume
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal t′* ∧
t = *Normal t′′* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs =
map (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∨
fst (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod-nest-call* ∧

xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*))
then show *?thesis*
proof
assume
a1:*fst* (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xsa*)) = *Normal t′* ∧
t = *Normal t″* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*)
then obtain *ys* **where** *p2-exec*:(*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift-catch P2*) *xsa* @
(*P2*, *snd* (((*P1*, *t*) # *xsa*) ! *length xsa*)) # *ys*
by *fastforce*
from *a1* **obtain** *t1* **where** *t-normal*: *t*=*Normal t1*
using *env-normal-s′-normal-s* **by** *blast*
have *f1*:*fst* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *t*)#*xsa*)) = *LanguageCon.com.Throw*
using *a1* **by** *fastforce*
from *a1* **have** *last-normal*: *snd* (*last* ((*P1*, *s*)#(*P1*, *t*) # *xsa*)) = *Normal t′*
by *fastforce*
then have *p2-long-exec*: (*n*,Γ, (*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *s*)#*xsa*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*P*, *t*)#*xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @
(*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *s*)#*xsa*))) # *ys* **using** *p2-exec*
by (*simp add*: *lift-catch-def local.Catch*)
thus *?thesis* **using** *a1 f1 last-normal all-step eq-P-Catch*
by (*clarify*, *metis* (*no-types*) *list.size*(*4*) *not-step-c-env step-e*)
next
assume
as1:*fst* (((*P1*, *t*) # *xsa*) ! *length xsa*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift-catch P2*) *xsa* @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*))
then obtain *ys* **where** *p1*:(*n*,Γ,(*Skip*,*snd*(*last* ((*P1*, *t*)#*xsa*)))#*ys*) ∈ *cptn-mod-nest-call* ∧
(*P*, *t*)#*xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @
((*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*)))#*ys*)
proof −
assume *a1*: ⋀*ys*. (*n*,Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*P*, *t*) # *xs* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*) @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P1*, *t*) # *xsa*))) # *ys* ⟹
thesis

have (*LanguageCon.com.Catch P1 P2*, *t*) # *map* (*lift-catch P2*) *xsa* = *map* (*lift-catch P2*) ((*P1*, *t*) # *xsa*)
by (*simp add*: *lift-catch-def*)
thus *?thesis*
using *a1 as1 eq-P-Catch* **by** *moura*
qed
from *as1* **have** *p2*: *fst* (((*P1*, *s*)#(*P1*, *t*) # *xsa*) ! *length* ((*P1*, *t*) #*xsa*)) = *LanguageCon.com.Skip*
by *fastforce*
thus *?thesis* **using** *p1 all-step eq-P-Catch* **by** *fastforce*
qed
qed
qed (*auto*)
qed(*force*+)

definition *seq-cond-nest*
where
seq-cond-nest zs Q xs P s s'' s' Γ n ≡ (*zs*=(*map* (*lift Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*)))) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* ∧ *s*=*Normal s''*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*)))))

lemma *div-seq-nest*: **assumes** *cptn-m*:(*n*,Γ,*list*) ∈ *cptn-mod-nest-call*
shows (∀ *s P Q zs*. *list*=(*Seq P Q*, *s*)#*zs* ⟶
(∃ *xs s' s''*.
(*n*,Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
seq-cond-nest zs Q xs P s s'' s' Γ n))

unfolding *seq-cond-nest-def*
using *cptn-m*
proof (*induct rule*: *cptn-mod-nest-call.induct*)
case (*CptnModNestOne* Γ *P s*)
thus *?case* **using** *cptn-mod-nest-call.CptnModNestOne*
by *blast*
next
case (*CptnModNestSkip* Γ *P s t n xs*)
from *CptnModNestSkip.hyps*
have *step*: $\Gamma\vdash_c$ (*P*, *s*) → (*Skip*, *t*) **by** *auto*
from *CptnModNestSkip.hyps*
have *noskip*: ~(*P*=*Skip*) **using** *stepc-elim-cases*(*1*) **by** *blast*
have *x*: ∀ *c c1 c2*. *redex c* = *Seq c1 c2* ⟹ *False*
using *redex-not-Seq* **by** *blast*

from *CptnModNestSkip.hyps*
have *in-cptn-mod*: $(n,\Gamma, (Skip, t) \# xs) \in$ *cptn-mod-nest-call* **by** *auto*
then show *?case* **using** *CptnModNestSkip.hyps*(*2*) *SmallStepCon.redex-not-Seq*
by *blast*
next
case (*CptnModNestThrow* Γ *P s t xs*)
from *CptnModNestThrow.hyps*
have *step*: $\Gamma\vdash_c (P, s) \rightarrow (Throw, t)$ **by** *auto*
moreover from *CptnModNestThrow.hyps*
have *no-seq*: $\forall p1\ p2.\ \neg(P=Seq\ p1\ p2)$ **using** *CptnModNestThrow.hyps*(*2*) *redex-not-Seq*
by *auto*
ultimately show *?case* **by** *auto*
next
case (*CptnModNestCondT* Γ *P0 s ys b P1*)
thus *?case* **by** *auto*
next
case (*CptnModNestCondF* Γ *P0 s ys b P1*)
thus *?case* **by** *auto*
next
case (*CptnModNestSeq1 n* Γ *P0 s xs zs P1*) **thus** *?case*
by *blast*
next
case (*CptnModNestSeq2 n* Γ *P0 s xs P1 ys zs*)
from *CptnModNestSeq2.hyps*(*3*) *last-length* **have** *last*:*fst* $(((P0, s) \# xs) ! length$ $xs) = Skip$
by (*simp add*: *last-length*)
have *P0cptn*:$(n,\Gamma, (P0, s) \# xs) \in$ *cptn-mod-nest-call* **by** *fact*
from *CptnModNestSeq2.hyps*(*4*) **have** *P1cptn*:$(n,\Gamma, (P1, snd\ (((P0, s) \# xs) !$ $length\ xs)) \# ys) \in$ *cptn-mod-nest-call*
by (*simp add*: *last-length*)
then have $zs = map\ (lift\ P1)\ xs\ @\ (P1, snd\ (last\ ((P0, s) \# xs))) \# ys$ **by** (*simp add*:*CptnModNestSeq2.hyps*)
show *?case*
proof −{
fix *sa P Q zsa*
assume *eq*:$(Seq\ P0\ P1, s) \# zs = (Seq\ P\ Q, sa) \# zsa$
then have $P0 = P \wedge P1 = Q \wedge s=sa \wedge zs=zsa$ **by** *auto*
have $last\ ((P0, s) \# xs) = ((P, sa) \# xs) !\ length\ xs$
by (*simp add*: ‹$P0 = P \wedge P1 = Q \wedge s = sa \wedge zs = zsa$› *last-length*)
then have $zsa = map\ (lift\ Q)\ xs\ @\ (Q, snd\ (((P, sa) \# xs) !\ length\ xs)) \# ys$
using ‹$P0 = P \wedge P1 = Q \wedge s = sa \wedge zs = zsa$› ‹$zs = map\ (lift\ P1)\ xs\ @$ $(P1, snd\ (last\ ((P0, s) \# xs))) \# ys$›
by *force*
then have $(\exists xs\ s'\ s''.\ (n,\Gamma, (P, sa) \# xs) \in$ *cptn-mod-nest-call* $\wedge$
$(zsa = map\ (lift\ Q)\ xs\ \vee$
$fst\ (((P, sa) \# xs) !\ length\ xs) = Skip\ \wedge$
$(\exists ys.\ (n,\Gamma, (Q, snd\ (((P, sa) \# xs) !\ length\ xs)) \# ys) \in$ *cptn-mod-nest-call* $\wedge$
$zsa = map\ (lift\ Q)\ xs\ @\ (Q, snd\ (((P, sa) \# xs) !\ length$

xs)) # *ys*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal s'* ∧ *s*=*Normal s''*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*))))))

using *P0cptn P1cptn* ‹*P0* = *P* ∧ *P1* = *Q* ∧ *s* = *sa* ∧ *zs* = *zsa*› *last*
by *blast*
}
thus *?case* **by** *auto* **qed**
next
case (*CptnModNestSeq3 n* Γ *P0 s xs s' ys zs P1*)
from *CptnModNestSeq3*.*hyps*(*3*)
have *last*:*fst* (((*P0*, *Normal s*) # *xs*) ! *length xs*) = *Throw*
by (*simp add*: *last-length*)
have *P0cptn*:(*n*,Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* **by** *fact*
from *CptnModNestSeq3*.*hyps*(*4*)
have *lastnormal*:*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s'*
by (*simp add*: *last-length*)
then have *zs* = *map* (*lift P1*) *xs* @ ((*Throw*, *Normal s'*)#*ys*) **by** (*simp add*:*CptnModNestSeq3*.*hyps*)
show *?case*
proof −**{**
fix *sa P Q zsa*
assume *eq*:(*Seq P0 P1*, *Normal s*) # *zs* = (*Seq P Q*, *Normal sa*) # *zsa*
then have *P0* =*P* ∧ *P1* = *Q* ∧ *Normal s*=*Normal sa* ∧ *zs*=*zsa* **by** *auto*
then have (*P0*, *Normal s*) = (*P*, *Normal sa*) **by** *auto*
have *last* ((*P0*, *Normal s*) # *xs*) = ((*P*, *Normal sa*) # *xs*) ! *length xs*
by (*simp add*: ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› *last-length*)
then have *zsa*:*zsa* = (*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*)
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› ‹*zs* = *map* (*lift P1*) *xs* @ ((*Throw*, *Normal s'*)#*ys*)›
by *force*
then have *a1*:(*n*,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* **using** *CptnModNestSeq3*.*hyps*(*5*) **by** *blast*
have (*P*, *Normal sa*::('*b*, '*c*) *xstate*) = (*P0*, *Normal s*)
using ‹*P0* = *P* ∧ *P1* = *Q* ∧ *Normal s* = *Normal sa* ∧ *zs* = *zsa*› **by** *auto*
then have (∃ *xs s'*. (*n*,Γ, (*P*, *Normal sa*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(*zsa* = *map* (*lift Q*) *xs* ∨
fst (((*P*,*Normal sa*) # *xs*) ! *length xs*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
zsa = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *Normal sa*) # *xs*) ! *length xs*)) # *ys*) ∨
((*fst*(((*P*, *Normal sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *Normal sa*)#*xs*)) = *Normal s'* ∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*))))))
using *P0cptn zsa a1 last lastnormal*

by *blast*
}
thus *?thesis* **by** *auto* **qed**
next
case (*CptnModNestEnv* Γ *P s t n zs*)
then have *step*:(*n*,Γ, (*P*, *t*) # *zs*) ∈ *cptn-mod-nest-call* **by** *auto*
have *step-e*: $\Gamma\vdash_c$ (*P*, *s*) $\rightarrow_e$ (*P*, *t*) **using** *CptnModNestEnv* **by** *auto*
show *?case*
proof (*cases P*)
case (*Seq P1 P2*)
then have *eq-P*:(*P*, *t*) # *zs* = (*LanguageCon.com.Seq P1 P2*, *t*) # *zs* **by** *auto*
then obtain *xs t′ t″* **where**
p1:(*n*,Γ, (*P1*, *t*) # *xs*) ∈ *cptn-mod-nest-call* **and** *p2*:
(*zs* = *map* (*lift P2*) *xs* ∨
fst (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
zs =
map (*lift P2*) *xs* @
(*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∨
fst (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xs*)) = *Normal t′* ∧
t = *Normal t″* ∧ (∃ *ys*. (*n*,Γ,(*Throw*,*Normal t′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zs =
map (*lift P2*) *xs* @
((*LanguageCon.com.Throw*, *Normal t′*)#*ys*)))
using *CptnModNestEnv*(*3*) **by** *auto*
have *all-step*:(*n*,Γ, (*P1*, *s*)#((*P1*, *t*) # *xs*)) ∈ *cptn-mod-nest-call*
using *p1 Env Env-n cptn-mod-nest-call.CptnModNestEnv env-normal-s step-e*
proof −
have *SmallStepCon.redex P* = *SmallStepCon.redex P1*
by (*metis SmallStepCon.redex.simps*(*4*) *local.Seq*)
then show *?thesis*
by (*metis* (*no-types*) *Env Env-n cptn-mod-nest-call.CptnModNestEnv env-normal-s p1 step-e*)
qed
show *?thesis* **using** *p2*
proof
assume *zs* = *map* (*lift P2*) *xs*
have (*P*, *t*) # *zs* = *map* (*lift P2*) ((*P1*, *t*) # *xs*)
by (*simp add*: ‹*zs* = *map* (*lift P2*) *xs*› *lift-def local.Seq*)
thus *?thesis* **using** *all-step eq-P* **by** *fastforce*
next
assume
fst (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
zs = *map* (*lift P2*) *xs* @ (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∨

fst (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xs*)) = *Normal t*′ ∧
t = *Normal t*′′ ∧ (∃ *ys*. (*n*,Γ,(*Throw*,*Normal t*′)#*ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift P2*) *xs* @ ((*LanguageCon.com.Throw*, *Normal t*′)#*ys*))
then show *?thesis*
proof
assume
a1:*fst* (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∈
cptn-mod-nest-call ∧
zs = *map* (*lift P2*) *xs* @ (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*)
from *a1* **obtain** *ys* **where**
p2-exec:(*n*,Γ, (*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*) ∈
cptn-mod-nest-call ∧
zs = *map* (*lift P2*) *xs* @
(*P2*, *snd* (((*P1*, *t*) # *xs*) ! *length xs*)) # *ys*
by *auto*
have *f1*:*fst* (((*P1*, *s*)#(*P1*, *t*) # *xs*) ! *length* ((*P1*, *t*)#*xs*)) =
LanguageCon.com.Skip
using *a1* **by** *fastforce*
then have *p2-long-exec*:
(*n*,Γ, (*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xs*) ! *length* ((*P1*, *t*)#*xs*))) #
ys) ∈ *cptn-mod-nest-call* ∧
(*P*, *t*)#*zs* = *map* (*lift P2*) ((*P1*, *t*) # *xs*) @
(*P2*, *snd* (((*P1*, *s*)#(*P1*, *t*) # *xs*) ! *length* ((*P1*, *t*)#*xs*))) # *ys*
using *p2-exec* **by** (*simp add*: *lift-def local.Seq*)
thus *?thesis* **using** *a1 f1 all-step eq-P* **by** *blast*
next
assume
a1:*fst* (((*P1*, *t*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P1*, *t*) # *xs*)) = *Normal t*′ ∧ *t* = *Normal t*′′ ∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal t*′)#*ys*) ∈ *cptn-mod-nest-call* ∧
zs = *map* (*lift P2*) *xs* @ ((*LanguageCon.com.Throw*, *Normal t*′)#*ys*))

then have *last-throw*:
fst (((*P1*, *s*)#(*P1*, *t*) # *xs*) ! *length* ((*P1*, *t*) #*xs*)) = *Language-Con.com.Throw*
by *fastforce*
from *a1* **have** *last-normal*: *snd* (*last* ((*P1*, *s*)#(*P1*, *t*) # *xs*)) = *Normal*
t′
by *fastforce*
have *seq-lift*:
(*LanguageCon.com.Seq P1 P2*, *t*) # *map* (*lift P2*) *xs* = *map* (*lift P2*)
((*P1*, *t*) # *xs*)
by (*simp add*: *a1 lift-def*)
thus *?thesis* **using** *a1 last-throw last-normal all-step eq-P*
by (*clarify*, *metis* (*no-types*, *lifting*) *append-Cons env-normal-s*′*-normal-s*
step-e)

qed
qed
qed (*auto*)
qed (*force*)+

lemma *map-lift-eq-xs-xs'*:*map* (*lift a*) *xs* = *map* (*lift a*) *xs'* $\Longrightarrow$ *xs*=*xs'*
proof (*induct xs arbitrary*: *xs'*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*Cons x xsa*)
then have *a0*:(*lift a*) *x* # *map* (*lift a*) *xsa* = *map* (*lift a*) (*x* # *xsa*)
by *fastforce*
also obtain *x' xsa'* **where** *xs'*:*xs'* = *x'*#*xsa'*
using *Cons* **by** *auto*
ultimately have *a1*:*map* (*lift a*) (*x* # *xsa*) =*map* (*lift a*) (*x'* # *xsa'*)
using *Cons* **by** *auto*
then have *xs*:*xsa*=*xsa'* **using** *a0 a1 Cons* **by** *fastforce*
then have (*lift a*) *x'* = (*lift a*) *x* **using** *a0 a1* **by** *auto*
then have *x'* = *x* **unfolding** *lift-def*
by (*metis* (*no-types*, *lifting*) *LanguageCon.com.inject*(*3*)
case-prod-beta old.prod.inject prod.collapse)
thus *?case* **using** *xs xs'* **by** *auto*
qed

lemma *map-lift-catch-eq-xs-xs'*:*map* (*lift-catch a*) *xs* = *map* (*lift-catch a*) *xs'* $\Longrightarrow$ *xs*=*xs'*
proof (*induct xs arbitrary*: *xs'*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*Cons x xsa*)
then have *a0*:(*lift-catch a*) *x* # *map* (*lift-catch a*) *xsa* = *map* (*lift-catch a*) (*x* # *xsa*)
by *auto*
also obtain *x' xsa'* **where** *xs'*:*xs'* = *x'*#*xsa'*
using *Cons* **by** *auto*
ultimately have *a1*:*map* (*lift-catch a*) (*x* # *xsa*) =*map* (*lift-catch a*) (*x'* # *xsa'*)
using *Cons* **by** *auto*
then have *xs*:*xsa*=*xsa'* **using** *a0 a1 Cons* **by** *fastforce*
then have (*lift-catch a*) *x'* = (*lift-catch a*) *x* **using** *a0 a1* **by** *auto*
then have *x'* = *x* **unfolding** *lift-catch-def*
by (*metis* (*no-types*, *lifting*) *LanguageCon.com.inject*(*9*)
case-prod-beta old.prod.inject prod.collapse)
thus *?case* **using** *xs xs'* **by** *auto*
qed

lemma *map-lift-all-seq*:
assumes *a0*:*zs*=*map* (*lift a*) *xs* **and**
a1:*i*<*length zs*
shows $\exists$ *b*. *fst* (*zs*!*i*) = *Seq b a*

using *a0 a1*
proof (*induct zs arbitrary*: *xs i*)
  case *Nil* **thus** *?case* **by** *auto*
next
  case (*Cons z1 zsa*) **thus** *?case* **unfolding** *lift-def*
  proof −
    assume *a1*: *z1* # *zsa* = *map* (λ*b*. *case b of* (*P*, *s*) ⇒ (*LanguageCon.com.Seq P a*, *s*)) *xs*
    have ∀ *p c*. ∃ *x*. ∀ *pa ca xa*.
        (*pa* ≠ (*ca*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*, *xa*::(′*a*, ′*c*) *xstate*) ∨ *ca* = *fst pa*) ∧
        ((*c*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*) ≠ *fst p* ∨ (*c*, *x*::(′*a*, ′*c*) *xstate*) = *p*)
      by *fastforce*
    then obtain *xx* :: (′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com* × (′*a*, ′*c*) *xstate* ⇒ (′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com* ⇒ (′*a*, ′*c*) *xstate* **where**
      ⋀*p c x ca pa*. (*p* ≠ (*c*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*, *x*::(′*a*, ′*c*) *xstate*) ∨ *c* = *fst p*) ∧ (*ca* ≠ *fst pa* ∨ (*ca*, *xx pa ca*) = *pa*)
      by (*metis* (*full-types*))
    then show *?thesis*
      using *a1* ‹*i* < *length* (*z1* # *zsa*)›
      by (*simp add*: *Cons.hyps Cons.prems*(*1*) *case-prod-beta′*)
  qed
qed

lemma *map-lift-catch-all-catch*:
 assumes *a0*:*zs*=*map* (*lift-catch a*) *xs* **and**
     a1:*i*<*length zs*
 shows ∃ *b*. *fst* (*zs*!*i*) = *Catch b a*
using *a0 a1*
proof (*induct zs arbitrary*: *xs i*)
  case *Nil* **thus** *?case* **by** *auto*
next
  case (*Cons z1 zsa*) **thus** *?case* **unfolding** *lift-catch-def*
  proof −
   assume *a1*: *z1* # *zsa* = *map* (λ*b*. *case b of* (*P*, *s*) ⇒ (*LanguageCon.com.Catch P a*, *s*)) *xs*
    have ∀ *p c*. ∃ *x*. ∀ *pa ca xa*.
        (*pa* ≠ (*ca*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*, *xa*::(′*a*, ′*c*) *xstate*) ∨ *ca* = *fst pa*) ∧
        ((*c*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*) ≠ *fst p* ∨ (*c*, *x*::(′*a*, ′*c*) *xstate*) = *p*)
      by *fastforce*
    then obtain *xx* :: (′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com* × (′*a*, ′*c*) *xstate* ⇒ (′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com* ⇒ (′*a*, ′*c*) *xstate* **where**
      ⋀*p c x ca pa*. (*p* ≠ (*c*::(′*a*, ′*b*, ′*c*, ′*d*) *LanguageCon.com*, *x*::(′*a*, ′*c*) *xstate*) ∨ *c* = *fst p*) ∧ (*ca* ≠ *fst pa* ∨ (*ca*, *xx pa ca*) = *pa*)
      by (*metis* (*full-types*))
    then show *?thesis*

using *a1* ‹*i* < *length* (*z1* # *zsa*)›
by (*simp add*: *Cons.hyps Cons.prems*(*1*) *case-prod-beta'*)
qed
qed

lemma *map-lift-some-eq-pos*:
assumes *a0*:*map* (*lift P*) *xs* @ (*P1*, *s1*)#*ys* =
map (*lift P*) *xs'*@ (*P2*, *s2*)#*ys'* **and**
a1:∀ *p0*. *P1*≠*Seq p0 P* **and**
a2:∀ *p0*. *P2*≠*Seq p0 P*
shows *length xs* = *length xs'*
proof −
{**assume** *ass*:*length xs* ≠ *length xs'*
{ **assume** *ass*:*length xs* < *length xs'*
then have *False* **using** *a0* *map-lift-all-seq a1 a2*
by (*metis* (*no-types*, *lifting*) *fst-conv length-map nth-append nth-append-length*)
}**note** *l*=*this*
{ **assume** *ass*:*length xs* > *length xs'*
then have *False* **using** *a0* *map-lift-all-seq a1 a2*
by (*metis* (*no-types*, *lifting*) *fst-conv length-map nth-append nth-append-length*)
} **then have** *False* **using** *l ass* **by** *fastforce*
}
thus *?thesis* **by** *auto*
qed

lemma *map-lift-some-eq*:
assumes *a0*:*map* (*lift P*) *xs* @ (*P1*, *s1*)#*ys* =
map (*lift P*) *xs'*@ (*P2*, *s2*)#*ys'* **and**
a1:∀ *p0*. *P1*≠*Seq p0 P* **and**
a2:∀ *p0*. *P2*≠*Seq p0 P*
shows *xs'* = *xs* ∧ *ys* = *ys'*
proof −
have *length xs* = *length xs'* **using** *a0 map-lift-some-eq-pos a1 a2* **by** *blast*
also have *xs'* = *xs* **using** *a0 assms calculation map-lift-eq-xs-xs'* **by** *fastforce*
ultimately show *?thesis* **using** *a0* **by** *fastforce*
qed

lemma *map-lift-catch-some-eq-pos*:
assumes *a0*:*map* (*lift-catch P*) *xs* @ (*P1*, *s1*)#*ys* =
map (*lift-catch P*) *xs'*@ (*P2*, *s2*)#*ys'* **and**
a1:∀ *p0*. *P1*≠*Catch p0 P* **and**
a2:∀ *p0*. *P2*≠*Catch p0 P*
shows *length xs* = *length xs'*
proof −
{**assume** *ass*:*length xs* ≠ *length xs'*
{ **assume** *ass*:*length xs* < *length xs'*
then have *False* **using** *a0* *map-lift-catch-all-catch a1 a2*
by (*metis* (*no-types*, *lifting*) *fst-conv length-map nth-append nth-append-length*)
}**note** *l*=*this*

{ **assume** *ass*:*length xs* > *length xs'*
then have *False* **using** *a0 map-lift-catch-all-catch a1 a2*
by (*metis* (*no-types*, *lifting*) *fst-conv length-map nth-append nth-append-length*)
} **then have** *False* **using** *l ass* **by** *fastforce*
}
thus *?thesis* **by** *auto*
qed

lemma *map-lift-catch-some-eq*:
assumes *a0*:*map* (*lift-catch P*) *xs* @ (*P1*, *s1*)#*ys* =
map (*lift-catch P*) *xs'*@ (*P2*, *s2*)#*ys'* **and**
a1:∀ *p0*. *P1*≠*Catch p0 P* **and**
a2:∀ *p0*. *P2*≠*Catch p0 P*
shows *xs'* = *xs* ∧ *ys* = *ys'*
proof −
have *length xs* = *length xs'* **using** *a0 map-lift-catch-some-eq-pos a1 a2* **by** *blast*
also have *xs'* = *xs* **using** *a0 assms calculation map-lift-catch-eq-xs-xs'* **by** *fastforce*
ultimately show *?thesis* **using** *a0* **by** *fastforce*
qed

lemma *Seq-P-Not-finish*:
assumes
a0:*zs* = *map* (*lift Q*) *xs* **and**
a1:(*m*, Γ,(*LanguageCon.com.Seq P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call* **and**
a2:*seq-cond-nest zs Q xs' P s s'' s'* Γ *m*
shows *xs*=*xs'*
using *a2* **unfolding** *seq-cond-nest-def*
proof
assume *zs*= *map* (*lift Q*) *xs'*
then have *map* (*lift Q*) *xs'* =
map (*lift Q*) *xs* **using** *a0* **by** *auto*
thus *?thesis* **using** *map-lift-eq-xs-xs'* **by** *fastforce*
next
assume
ass:*fst* (((*P*, *s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift Q*) *xs'* @ (*Q*, *snd* (((*P*, *s*) # *xs'*) ! *length xs'*)) # *ys*) ∨
fst (((*P*, *s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *s*) # *xs'*)) = *Normal s'* ∧
s = *Normal s''* ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Throw*, *Normal s'*) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift Q*) *xs'* @ (*LanguageCon.com.Throw*, *Normal s'*) # *ys*)
{**assume**
ass:*fst* (((*P*, *s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call*
∧

zs = map (lift Q) xs' @ (Q, snd (((P, s) # xs') ! length xs')) # ys)
then obtain *ys* **where**
zs:zs = map (lift Q) xs' @ (Q, snd (((P, s) # xs') ! length xs')) # ys
by *auto*
then have *zs-while:fst (zs!(length (map (lift Q) xs'))) =*
Q **by** *(metis fstI nth-append-length)*
have *length zs = length (map (lift Q) xs' @*
(Q, snd (((P, s) # xs') ! length xs')) # ys)
using *zs* **by** *auto*
then have *(length (map (lift Q) xs')) <*
length zs **by** *auto*
then have *?thesis* **using** *a0 zs-while map-lift-all-seq*
using *seq-and-if-not-eq(4)* **by** *fastforce*
}note *l = this*
{assume *ass:fst (((P, s) # xs') ! length xs') = LanguageCon.com.Throw* $\wedge$
snd (last ((P, s) # xs')) = Normal s' $\wedge$
s = Normal s'' $\wedge$
($\exists$ *ys. (m,* Γ*, (LanguageCon.com.Throw, Normal s') # ys)* $\in$ *cptn-mod-nest-call*
$\wedge$
zs = map (lift Q) xs' @ (LanguageCon.com.Throw, Normal s') # ys)
then obtain *ys* **where**
zs:zs = map (lift Q) xs' @
(LanguageCon.com.Throw, Normal s') # ys **by** *auto*
then have *zs-while:*
fst (zs!(length (map (lift Q) xs'))) = Throw **by** *(metis fstI nth-append-length)*

have *length zs = length (map (lift Q) xs' @(LanguageCon.com.Throw, Normal s') # ys)*
using *zs* **by** *auto*
then have *(length (map (lift Q) xs')) <*
length zs **by** *auto*
then have *?thesis* **using** *a0 zs-while map-lift-all-seq*
using *seq-and-if-not-eq(4)* **by** *fastforce*
} thus *?thesis* **using** *l ass* **by** *auto*
qed

lemma *Seq-P-Ends-Normal*:
assumes
a0:zs = map (lift Q) xs @ (Q, snd (last ((P, s) # xs))) # ys **and**
a0':fst (last ((P, s) # xs)) = Skip **and**
a1:(m, Γ*,(LanguageCon.com.Seq P Q, s) # zs)* $\in$ *cptn-mod-nest-call* **and**
a2:seq-cond-nest zs Q xs' P s s'' s' Γ *m*
shows *xs=xs'* $\wedge$ *(m,*Γ*,(Q, snd(((P, s)#xs)!length xs))#ys)* $\in$ *cptn-mod-nest-call*
using *a2* **unfolding** *seq-cond-nest-def*
proof
assume *ass:zs= map (lift Q) xs'*
then have *map (lift Q) xs' =*
map (lift Q) xs @ (Q, snd (last ((P, s) # xs))) # ys **using** *a0* **by**
auto

then have *zs-while*:*fst* (*zs*!(*length* (*map* (*lift* *Q*) *xs*))) = *Q*
by (*metis a0 fstI nth-append-length*)
also have *length zs* =
length (*map* (*lift* *Q*) *xs* @ (*Q*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*)
using *a0* **by** *auto*
then have (*length* (*map* (*lift* *Q*) *xs*)) < *length zs* **by** *auto*
then show *?thesis* **using** *ass zs-while map-lift-all-seq*
using *seq-and-if-not-eq*(*4*)
by *metis*
next
assume
ass:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*) ∨
fst (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *s*) # *xs*′)) = *Normal s*′ ∧
s = *Normal s*′′ ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Throw*, *Normal s*′) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift* *Q*) *xs*′ @ (*LanguageCon.com.Throw*, *Normal s*′) # *ys*)
{**assume**
ass:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*)
then obtain *ys*′ **where**
zs:*zs* = *map* (*lift* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*′ ∧
(*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*′) ∈ *cptn-mod-nest-call*

by *auto*
then have *?thesis*
using *map-lift-some-eq*[*of Q xs Q - ys xs*′ *Q - ys*′]
zs a0 seq-and-if-not-eq(*4*)[*of Q*]
by *auto*
}**note** *l* = *this*
{**assume** *ass*:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *s*) # *xs*′)) = *Normal s*′ ∧
s = *Normal s*′′ ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Throw*, *Normal s*′) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift* *Q*) *xs*′ @ (*LanguageCon.com.Throw*, *Normal s*′) # *ys*)
then obtain *ys*′ **where**
zs:*zs* = *map* (*lift* *Q*) *xs*′ @ (*LanguageCon.com.Throw*, *Normal s*′) # *ys*′ ∧
(*m*, Γ, (*LanguageCon.com.Throw*, *Normal s*′) # *ys*′) ∈ *cptn-mod-nest-call*

by *auto*
then have *zs-while*:
fst (*zs*!(*length* (*map* (*lift* *Q*) *xs*′))) = *Throw* **by** (*metis fstI nth-append-length*)

have *False*
by (*metis* (*no-types*) *LanguageCon.com.distinct*(*17*)
LanguageCon.com.distinct(*71*)
a0 a0′ ass last-length
map-lift-some-eq seq-and-if-not-eq(*4*) *zs*)
then have *?thesis*
by *metis*
} **thus** *?thesis* **using** *l ass* **by** *auto*
qed

lemma *Seq-P-Ends-Abort*:
assumes
a0:*zs* = *map* (*lift Q*) *xs* @ (*Throw*, *Normal s′*) # *ys* **and**
a0′:*fst* (*last* ((*P*, *Normal s*) # *xs*)) = *Throw* **and**
a0′′:*snd*(*last* ((*P*, *Normal s*) # *xs*)) = *Normal s′* **and**
a1:(*m*, Γ,(*LanguageCon.com.Seq P Q*, *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
and
a2:*seq-cond-nest zs Q xs′ P* (*Normal s*) *ns′′ ns′* Γ *m*
shows *xs*=*xs′* ∧ (*m*,Γ,(*Throw*,*Normal s′*)#*ys*) ∈ *cptn-mod-nest-call*
using *a2* **unfolding** *seq-cond-nest-def*
proof
assume *ass*:*zs*= *map* (*lift Q*) *xs′*
then have *map* (*lift Q*) *xs′* =
map (*lift Q*) *xs* @ (*Throw*, *Normal s′*) # *ys* **using** *a0* **by** *auto*
then have *zs-while*:*fst* (*zs*!(*length* (*map* (*lift Q*) *xs*))) = *Throw*
by (*metis a0 fstI nth-append-length*)
also have *length zs* =
length (*map* (*lift Q*) *xs* @ (*Throw*, *Normal s′*) # *ys*)
using *a0* **by** *auto*
then have (*length* (*map* (*lift Q*) *xs*)) < *length zs* **by** *auto*
then show *?thesis* **using** *ass zs-while map-lift-all-seq*
by (*metis* (*no-types*) *LanguageCon.com.simps*(*82*))
next
assume
ass:*fst* (((*P*, *Normal s*) # *xs′*) ! *length xs′*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *Normal s*) # *xs′*) ! *length xs′*)) # *ys*)
∈ *cptn-mod-nest-call* ∧
zs = *map* (*lift Q*) *xs′* @
(*Q*, *snd* (((*P*, *Normal s*) # *xs′*) ! *length xs′*)) # *ys*) ∨
fst (((*P*, *Normal s*) # *xs′*) ! *length xs′*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *Normal s*) # *xs′*)) = *Normal ns′* ∧
Normal s = *Normal ns′′* ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Throw*, *Normal ns′*) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift Q*) *xs′* @ (*LanguageCon.com.Throw*, *Normal ns′*) # *ys*)
{assume
ass:*fst* (((*P*, *Normal s*) # *xs′*) ! *length xs′*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *Normal s*) # *xs′*) ! *length xs′*)) # *ys*)

$\in$ *cptn-mod-nest-call* $\wedge$
zs = *map* (*lift Q*) *xs'* @
(*Q*, *snd* (((*P*, *Normal s*) # *xs'*) ! *length xs'*)) # *ys*)
then obtain *ys'* **where**
zs:(*m*, Γ, (*Q*, *snd* (((*P*, *Normal s*) # *xs'*) ! *length xs'*)) # *ys'*)
$\in$ *cptn-mod-nest-call* $\wedge$
zs = *map* (*lift Q*) *xs'* @
(*Q*, *snd* (((*P*, *Normal s*) # *xs'*) ! *length xs'*)) # *ys'*
by *auto*
then have *?thesis*
using *a0 seq-and-if-not-eq*(*4*)[*of Q*]
by (*metis LanguageCon.com.distinct*(*17*) *LanguageCon.com.distinct*(*71*)
a0' ass last-length map-lift-some-eq)
}**note** *l* = *this*
{**assume** *ass*:*fst* (((*P*, *Normal s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Throw*
$\wedge$
snd (*last* ((*P*, *Normal s*) # *xs'*)) = *Normal ns'* $\wedge$
Normal s = *Normal ns''* $\wedge$
($\exists$ *ys*. (*m*, Γ, (*LanguageCon.com.Throw*, *Normal ns'*) # *ys*) $\in$ *cptn-mod-nest-call*
$\wedge$
zs = *map* (*lift Q*) *xs'* @ (*LanguageCon.com.Throw*, *Normal ns'*) # *ys*)
then obtain *ys'* **where**
zs:(*m*, Γ, (*LanguageCon.com.Throw*, *Normal ns'*) # *ys'*) $\in$ *cptn-mod-nest-call*
$\wedge$
zs = *map* (*lift Q*) *xs'* @ (*LanguageCon.com.Throw*, *Normal ns'*) # *ys'*
by *auto*
then have *zs-while*:
fst (*zs*!(*length* (*map* (*lift Q*) *xs'*))) = *Throw*
by (*metis fstI nth-append-length*)
then have *?thesis* **using** *a0 ass map-lift-some-eq* **by** *blast*
} **thus** *?thesis* **using** *l ass* **by** *auto*
qed

lemma *Catch-P-Not-finish*:
assumes
a0:*zs* = *map* (*lift-catch Q*) *xs* **and**
a1:*catch-cond-nest zs Q xs' P s s'' s'* Γ *m*
shows *xs*=*xs'*
using *a1* **unfolding** *catch-cond-nest-def*
proof
assume *zs*= *map* (*lift-catch Q*) *xs'*
then have *map* (*lift-catch Q*) *xs'* =
map (*lift-catch Q*) *xs* **using** *a0* **by** *auto*
thus *?thesis* **using** *map-lift-catch-eq-xs-xs'* **by** *fastforce*
next
assume
ass:
fst (((*P*, *s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P*, *s*) # *xs'*)) = *Normal s'* $\wedge$

$s = Normal\ s'' \wedge$
$(\exists\, ys.\ (m, \Gamma, (Q, snd\ (((P, s) \# xs') ! length\ xs')) \# ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (Q, snd\ (((P, s) \# xs') ! length\ xs')) \# ys)$
$\vee$
$fst\ (((P, s) \# xs') ! length\ xs') = LanguageCon.com.Skip \wedge$
$(\exists\, ys.\ (m, \Gamma, (LanguageCon.com.Skip, snd\ (last\ ((P, s) \# xs'))) \# ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (LanguageCon.com.Skip, snd\ (last\ ((P, s)$
$\# xs'))) \# ys)$
{assume
ass:$fst\ (((P, s) \# xs') ! length\ xs') = LanguageCon.com.Skip \wedge$
$(\exists\, ys.\ (m, \Gamma, (LanguageCon.com.Skip, snd\ (last\ ((P, s) \# xs'))) \# ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (LanguageCon.com.Skip, snd\ (last\ ((P, s)$
$\# xs'))) \# ys)$
then obtain *ys* **where**
zs:$(m, \Gamma, (LanguageCon.com.Skip, snd\ (last\ ((P, s) \# xs'))) \# ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (LanguageCon.com.Skip, snd\ (last\ ((P, s)$
$\# xs'))) \# ys$
by *auto*
then have *zs-while*:$fst\ (zs!(length\ (map\ (lift\text{-}catch\ Q)\ xs'))) = Skip$
by (*metis fstI nth-append-length*)
have $length\ zs = length\ (map\ (lift\ Q)\ xs' @$
$(Q, snd\ (((P, s) \# xs') ! length\ xs')) \# ys)$
using *zs* **by** *auto*
then have $(length\ (map\ (lift\ Q)\ xs')) <$
$length\ zs$ **by** *auto*
then have *?thesis* **using** *a0 zs-while map-lift-catch-all-catch*
using *seq-and-if-not-eq(12)* **by** *fastforce*
}note $l = this$
{assume *ass*:$fst\ (((P, s) \# xs') ! length\ xs') = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P, s) \# xs')) = Normal\ s' \wedge$
$s = Normal\ s'' \wedge$
$(\exists\, ys.\ (m, \Gamma, (Q, snd\ (((P, s) \# xs') ! length\ xs')) \# ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (Q, snd\ (((P, s) \# xs') ! length\ xs')) \# ys)$
then obtain *ys* **where**
zs:$zs = map\ (lift\text{-}catch\ Q)\ xs' @ (Q, snd\ (((P, s) \# xs') ! length\ xs'))$
$\# ys$ **by** *auto*
then have *zs-while*:
$fst\ (zs!(length\ (map\ (lift\ Q)\ xs'))) = Q$
by (*metis (no-types) eq-fst-iff length-map nth-append-length zs*)
have $length\ zs = length\ (map\ (lift\ Q)\ xs' @(LanguageCon.com.Throw,$
$Normal\ s') \# ys)$
using *zs* **by** *auto*
then have $(length\ (map\ (lift\ Q)\ xs')) <$
$length\ zs$ **by** *auto*

then have *?thesis* **using** *a0 zs-while map-lift-catch-all-catch*
by *fastforce*
} **thus** *?thesis* **using** *l ass* **by** *auto*
qed

lemma *Catch-P-Ends-Normal*:
assumes
a0:$zs = map\ (lift\text{-}catch\ Q)\ xs\ @\ (Q,\ snd\ (last\ ((P,\ Normal\ s)\ \#\ xs)))\ \#\ ys$
and
a0':$fst\ (last\ ((P,\ Normal\ s)\ \#\ xs)) = Throw$ **and**
a0'':$snd\ (last\ ((P,\ Normal\ s)\ \#\ xs)) = Normal\ s'$ **and**
a1:$catch\text{-}cond\text{-}nest\ zs\ Q\ xs'\ P\ (Normal\ s)\ ns''\ ns'\ \Gamma\ m$
shows $xs=xs' \wedge (m,\Gamma,(Q,\ snd(((P,\ Normal\ s)\#xs)!length\ xs))\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
using *a1* **unfolding** *catch-cond-nest-def*
proof
assume *ass*:$zs= map\ (lift\text{-}catch\ Q)\ xs'$
then have $map\ (lift\text{-}catch\ Q)\ xs' =$
$map\ (lift\text{-}catch\ Q)\ xs\ @\ (Q,\ snd\ (last\ ((P,\ Normal\ s)\ \#\ xs)))\ \#\ ys$
using *a0* **by** *auto*
then have *zs-while*:$fst\ (zs!(length\ (map\ (lift\text{-}catch\ Q)\ xs))) = Q$
by (*metis a0 fstI nth-append-length*)
also have $length\ zs =$
$length\ (map\ (lift\text{-}catch\ Q)\ xs\ @\ (Q,\ snd\ (last\ ((P,\ Normal\ s)\ \#\ xs)))$
$\#\ ys)$
using *a0* **by** *auto*
then have $(length\ (map\ (lift\text{-}catch\ Q)\ xs)) < length\ zs$ **by** *auto*
then show *?thesis* **using** *ass zs-while map-lift-catch-all-catch*
using *seq-and-if-not-eq*(*12*)
by *metis*
next
assume
ass:$fst\ (((P,\ Normal\ s)\ \#\ xs')\ !\ length\ xs') = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P,\ Normal\ s)\ \#\ xs')) = Normal\ ns' \wedge$
$Normal\ s = Normal\ ns'' \wedge$
$(\exists ys.\ (m,\ \Gamma,\ (Q,\ snd\ (((P,\ Normal\ s)\ \#\ xs')\ !\ length\ xs'))\ \#\ ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs'\ @\ (Q,\ snd\ (((P,\ Normal\ s)\ \#\ xs')\ !\ length\ xs'))$
$\#\ ys) \vee$
$fst\ (((P,\ Normal\ s)\ \#\ xs')\ !\ length\ xs') = LanguageCon.com.Skip \wedge$
$(\exists ys.\ (m,\ \Gamma,\ (LanguageCon.com.Skip,\ snd\ (last\ ((P,\ Normal\ s)\ \#\ xs')))\ \#$
$ys) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs'\ @\ (LanguageCon.com.Skip,\ snd\ (last\ ((P,$
$Normal\ s)\ \#\ xs')))\ \#\ ys)$
{assume
ass:$fst\ (((P,\ Normal\ s)\ \#\ xs')\ !\ length\ xs') = LanguageCon.com.Skip \wedge$
$(\exists ys.\ (m,\ \Gamma,\ (LanguageCon.com.Skip,\ snd\ (last\ ((P,\ Normal\ s)\ \#\ xs')))\ \#$
$ys) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs = map\ (lift\text{-}catch\ Q)\ xs'\ @\ (LanguageCon.com.Skip,\ snd\ (last\ ((P,$
$Normal\ s)\ \#\ xs')))\ \#\ ys)$

then obtain *ys'* **where**
zs:(m, Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((P, *Normal s*) # *xs'*))) # *ys'*) $\in$ *cptn-mod-nest-call* $\wedge$
zs = *map* (*lift-catch Q*) *xs'* @ (*LanguageCon.com.Skip*, *snd* (*last* ((P, *Normal s*) # *xs'*))) # *ys'*
by *auto*
then have *?thesis*
using *map-lift-catch-some-eq*[*of Q xs Q - ys xs' Skip - ys'*]
zs a0 seq-and-if-not-eq(*12*)[*of Q*]
by (*metis LanguageCon.com.distinct*(*17*) *LanguageCon.com.distinct*(*19*) *a0' ass last-length*)
}**note** *l* = *this*
{**assume** *ass*:*fst* (((*P*, *Normal s*) # *xs'*) ! *length xs'*) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((P, *Normal s*) # *xs'*)) = *Normal ns'* $\wedge$
Normal s = *Normal ns''* $\wedge$
($\exists$ *ys*. (m, Γ, (Q, *snd* (((P, *Normal s*) # *xs'*) ! *length xs'*)) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
zs = *map* (*lift-catch Q*) *xs'* @ (Q, *snd* (((P, *Normal s*) # *xs'*) ! *length xs'*)) # *ys*)
then obtain *ys'* **where**
zs:(m, Γ, (Q, *snd* (((P, *Normal s*) # *xs'*) ! *length xs'*)) # *ys'*) $\in$ *cptn-mod-nest-call* $\wedge$
zs = *map* (*lift-catch Q*) *xs'* @ (Q, *snd* (((P, *Normal s*) # *xs'*) ! *length xs'*)) # *ys'*
by *auto*
then have *zs-while*:
fst (*zs*!(*length* (*map* (*lift-catch Q*) *xs'*))) = Q **by** (*metis fstI nth-append-length*)

then have *?thesis*
using *LanguageCon.com.distinct*(*17*) *LanguageCon.com.distinct*(*71*)
a0 a0' ass last-length map-lift-catch-some-eq[*of Q xs Q - ys xs' Q - ys'*]
seq-and-if-not-eq(*12*) *zs*
by *blast*
} **thus** *?thesis* **using** *l ass* **by** *auto*
qed

lemma *Catch-P-Ends-Skip*:
assumes
a0:*zs* = *map* (*lift-catch Q*) *xs* @ (*Skip*, *snd* (*last* ((P, s) # *xs*))) # *ys* **and**
a0':*fst* (*last* ((P,s) # *xs*)) = *Skip* **and**
a1:*catch-cond-nest zs Q xs' P s ns'' ns'* Γ m
shows *xs*=*xs'* $\wedge$ (m,Γ,(*Skip*,*snd*(*last* ((P,s) # *xs*)))#*ys*) $\in$ *cptn-mod-nest-call*
using *a1* **unfolding** *catch-cond-nest-def*
proof
assume *ass*:*zs*= *map* (*lift-catch Q*) *xs'*
then have *map* (*lift-catch Q*) *xs'* =
map (*lift-catch Q*) *xs* @ (*Skip*, *snd* (*last* ((P, s) # *xs*))) # *ys* **using**

a0 **by** *auto*
then have *zs-while*:*fst* (*zs*!(*length* (*map* (*lift-catch* *Q*) *xs*))) = *Skip*
by (*metis a0 fstI nth-append-length*)
also have *length zs* =
length (*map* (*lift-catch* *Q*) *xs* @ (*Skip*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*)
using *a0* **by** *auto*
then have (*length* (*map* (*lift-catch* *Q*) *xs*)) < *length zs* **by** *auto*
then show *?thesis* **using** *ass zs-while map-lift-catch-all-catch*
by (*metis LanguageCon.com.distinct*(*19*))
next
assume
ass:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *s*) # *xs*′)) = *Normal ns*′ ∧
s = *Normal ns*′′ ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*)
∨
fst (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P*, *s*) # *xs*′))) # *ys*) ∈
cptn-mod-nest-call ∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*LanguageCon.com.Skip*, *snd* (*last* ((*P*, *s*)
xs′))) # *ys*)
{**assume**
ass:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*m*, Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P*, *s*) # *xs*′))) # *ys*) ∈
cptn-mod-nest-call ∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*LanguageCon.com.Skip*, *snd* (*last* ((*P*, *s*)
xs′))) # *ys*)
then obtain *ys*′ **where**
zs:(*m*, Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P*, *s*) # *xs*′))) # *ys*′) ∈
cptn-mod-nest-call ∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*LanguageCon.com.Skip*, *snd* (*last* ((*P*,
s) # *xs*′))) # *ys*′
by *auto*
then have *?thesis*
using *a0 seq-and-if-not-eq*(*12*)[*of Q*] *a0*′ *ass last-length map-lift-catch-some-eq*
using *LanguageCon.com.distinct*(*19*) **by** *blast*
}**note** *l* = *this*
{**assume** *ass*:*fst* (((*P*, *s*) # *xs*′) ! *length xs*′) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P*, *s*) # *xs*′)) = *Normal ns*′ ∧
s = *Normal ns*′′ ∧
(∃ *ys*. (*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*)
then obtain *ys*′ **where**
zs:(*m*, Γ, (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*′) ∈ *cptn-mod-nest-call*
∧
zs = *map* (*lift-catch* *Q*) *xs*′ @ (*Q*, *snd* (((*P*, *s*) # *xs*′) ! *length xs*′)) # *ys*′

by *auto*
then have *zs-while*:
fst (*zs*!(*length* (*map* (*lift-catch Q*) *xs'*))) = *Q*
by (*metis fstI nth-append-length*)
then have *?thesis*
using *a0 seq-and-if-not-eq*(*12*)[*of Q*] *a0' ass last-length map-lift-catch-some-eq*
by (*metis LanguageCon.com.distinct*(*17*) *LanguageCon.com.distinct*(*19*))

} **thus** *?thesis* **using** *l ass* **by** *auto*
qed

lemma *not-func-redex-cptn-mod-nest-n'*:
assumes *a0*:$\Gamma\vdash_c$ (*P*,*s*) → (*Q*, *t*) **and**
a1:(*n*,Γ,(*Q*,*t*)#*xs*) ∈ *cptn-mod-nest-call* **and**
a2:(∀ *fn*. *redex P* ≠ *Call fn*) ∨
(*redex P* = *Call fn* ∧ Γ *fn* = *None*) ∨
(*redex P* = *Call fn* ∧ (∀ *sa*. *s*≠*Normal sa*))
shows (*n*,Γ,(*P*,*s*)#(*Q*,*t*)#*xs*) ∈ *cptn-mod-nest-call*
using *a0 a1 a2*
proof (*induct arbitrary*: *xs*)
case (*Basicc f s*)
thus *?case* **by** (*simp add*: *Basicc cptn-mod-nest-call.CptnModNestSkip stepc.Basicc*)
next
case (*Specc s t r*)
thus *?case* **by** (*simp add*: *Specc cptn-mod-nest-call.CptnModNestSkip stepc.Specc*)
next
case (*SpecStuckc s r*)
thus *?case* **by** (*simp add*: *SpecStuckc cptn-mod-nest-call.CptnModNestSkip stepc.SpecStuckc*)
next
case (*Guardc s g f c*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestGuard*)
next

case (*GuardFaultc s g f c*)
thus *?case* **by** (*simp add*: *GuardFaultc cptn-mod-nest-call.CptnModNestSkip stepc.GuardFaultc*)
next
case (*Seqc c1 s c1' s' c2*)
have *step*: $\Gamma\vdash_c$ (*c1*, *s*) → (*c1'*, *s'*) **by** (*simp add*: *Seqc.hyps*(*1*))
then have *nsc1*: *c1*≠*Skip* **using** *stepc-elim-cases*(*1*) **by** *blast*
have *assum*: (*n*, Γ, (*Seq c1' c2*, *s'*) # *xs*) ∈ *cptn-mod-nest-call* **using** *Seqc.prems* **by** *blast*
have *divseq*:(∀ *s P Q zs*. (*Seq c1' c2*, *s'*) # *xs*=(*Seq P Q*, *s*)#*zs* ⟶
(∃ *xs sv' sv''*. ((*n*,Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
(*zs*=(*map* (*lift Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call* ∧

zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length* *xs*))#*ys*)))) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal sv′* ∧ *s′*=*Normal sv′′*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal sv′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal sv′*)#*ys*))
))))

)) **using** *div-seq-nest* [*OF assum*] **unfolding** *seq-cond-nest-def* **by** *auto*
{**fix** *sa P Q zsa*
assume *ass*:(*Seq c1′ c2*, *s′*) # *xs* = (*Seq P Q*, *sa*) # *zsa*
then have *eqs*:*c1′* = *P* ∧ *c2* = *Q* ∧ *s′* = *sa* ∧ *xs* = *zsa* **by** *auto*
then have (∃ *xs sv′ sv′′*. (*n*,Γ, (*P*, *sa*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(*zsa* = *map* (*lift Q*) *xs* ∨
fst (((*P*, *sa*) # *xs*) ! *length xs*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
zsa = *map* (*lift Q*) *xs* @ (*Q*, *snd* (((*P*, *sa*) # *xs*) ! *length* *xs*)) # *ys*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal sv′* ∧ *s′*=*Normal sv′′*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal sv′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zsa=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal sv′*)#*ys*))))))
using *ass divseq* **by** *blast*
} **note** *conc*=*this* [*of c1′ c2 s′ xs*]
then obtain *xs′ sa′ sa′′*
where *split*:(*n*,Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
(*xs* = *map* (*lift c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift c2*) *xs′* @ (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length* *xs′*)) # *ys*) ∨
((*fst*(((*c1′*, *s′*)#*xs′*)!*length xs′*)=*Throw* ∧
snd(*last* ((*c1′*, *s′*)#*xs′*)) = *Normal sa′* ∧ *s′*=*Normal sa′′*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal sa′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift c2*) *xs′*)@((*Throw*,*Normal sa′*)#*ys*))
))) **by** *blast*
then have (*xs* = *map* (*lift c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift c2*) *xs′* @ (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length* *xs′*)) # *ys*) ∨
((*fst*(((*c1′*, *s′*)#*xs′*)!*length xs′*)=*Throw* ∧
snd(*last* ((*c1′*, *s′*)#*xs′*)) = *Normal sa′* ∧ *s′*=*Normal sa′′*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal sa′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift c2*) *xs′*)@((*Throw*,*Normal sa′*)#*ys*)))))

by *auto*
thus *?case*
proof{
assume *c1'nonf*:*xs* = *map* (*lift c2*) *xs'*
then have *c1'cptn*:(*n*,Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *split* **by** *blast*
then have *induct-step*: (*n*,Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod-nest-call*
using *Seqc.hyps*(*2*) *Seqc.prems*(*2*) *SmallStepCon.redex.simps*(*4*) **by** *auto*
then have (*Seq c1' c2*, *s'*)#*xs* = *map* (*lift c2*) ((*c1'*, *s'*)#*xs'*)
using *c1'nonf*
by (*simp add*: *lift-def*)
thus *?thesis*
using *c1'nonf c1'cptn induct-step* **by** (*auto simp add*: *CptnModNestSeq1*)
next
assume *fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∨
((*fst*(((*c1'*, *s'*)#*xs'*)!*length xs'*)=*Throw* ∧
snd(*last* ((*c1'*, *s'*)#*xs'*)) = *Normal sa'* ∧ *s'*=*Normal sa''*∧
(∃ *ys*. (*n*,Γ,(*Throw*,*Normal sa'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift c2*) *xs'*)@((*Throw*,*Normal sa'*)#*ys*))))
thus *?thesis*
proof
assume *assth*:*fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*)
then obtain *ys*
where *split'*:(*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*
by *auto*
then have *c1'cptn*:(*n*,Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *split* **by** *blast*
then have *induct-step*: (*n*,Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod-nest-call*
using *Seqc.hyps*(*2*) *Seqc.prems*(*2*) *SmallStepCon.redex.simps*(*4*) **by** *auto*

then have *seqmap*:(*Seq c1 c2*, *s*)#(*Seq c1' c2*, *s'*)#*xs* = *map* (*lift c2*) ((*c1*,*s*)#(*c1'*, *s'*)#*xs'*) @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*
using *split'* **by** (*simp add*: *lift-def*)
then have *lastc1*:*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*) = ((*c1'*, *s'*) # *xs'*) ! *length xs'*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Skip*
using *assth* **by** *fastforce*
thus *?thesis*

using *seqmap split′ cptn-mod-nest-call.CptnModNestSeq2*
induct-step lastc1 lastc1skip
by (*metis* (*no-types*) *Cons-lift-append*)
next
assume *assm*:((*fst*(((*c1′*, *s′*)#*xs′*)!*length xs′*)=*Throw* ∧
snd(*last* ((*c1′*, *s′*)#*xs′*)) = *Normal sa′* ∧ *s′*=*Normal sa″*∧
(∃ *ys*.(*n*,Γ,(*Throw*,*Normal sa′*)#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift c2*) *xs′*)@((*Throw*,*Normal sa′*)#*ys*))))
then have *s′eqsa″*: *s′*=*Normal sa″* **by** *auto*
then have *snormal*: ∃ *ns*. *s*=*Normal ns* **by** (*metis Seqc.hyps*(*1*) *step-Abrupt-prop step-Fault-prop step-Stuck-prop xstate.exhaust*)
then have *c1′cptn*:(*n*,Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *split* **by** *blast*
then have *induct-step*: (*n*,Γ, (*c1*, *s*) # (*c1′*, *s′*)#*xs′*) ∈ *cptn-mod-nest-call*
using *Seqc.hyps*(*2*) *Seqc.prems*(*2*) *SmallStepCon.redex.simps*(*4*) **by** *auto*
then obtain *ys* **where** *seqmap*:(*Seq c1′ c2*, *s′*)#*xs* = (*map* (*lift c2*) ((*c1′*, *s′*)#*xs′*))@((*Throw*,*Normal sa′*)#*ys*)
using *assm*
proof −
assume *a1*: ⋀*ys*. (*LanguageCon.com.Seq c1′ c2*, *s′*) # *xs* = *map* (*lift c2*) ((*c1′*, *s′*) # *xs′*) @ (*LanguageCon.com.Throw*, *Normal sa′*) # *ys* ⟹ *thesis*
have (*LanguageCon.com.Seq c1′ c2*, *Normal sa″*) # *map* (*lift c2*) *xs′* = *map* (*lift c2*) ((*c1′*, *s′*) # *xs′*)
by (*simp add*: *assm lift-def*)
thus *?thesis*
using *a1 assm* **by** *moura*
qed
then have *lastc1*:*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*) = ((*c1′*, *s′*) # *xs′*) ! *length xs′*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *Throw*
using *assm* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1′*, *s′*) # *xs′*)) = *Normal sa′*
using *assm* **by** *force*
thus *?thesis*
using *assm c1′cptn induct-step lastc1skip snormal seqmap s′eqsa″*
by (*auto simp add*:*cptn-mod-nest-call.CptnModNestSeq3*)
qed
}**qed**
next
case (*SeqSkipc c2 s xs*)
have *c2incptn*:(*n*,Γ, (*c2*, *s*) # *xs*) ∈ *cptn-mod-nest-call* **by** *fact*
then have *1*:(*n*,Γ, [(*Skip*, *s*)]) ∈ *cptn-mod-nest-call*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
then have *2*:*fst*(*last* ([(*Skip*, *s*)])) = *Skip* **by** *fastforce*
then have *3*:(*n*,Γ,(*c2*, *snd*(*last* [(*Skip*, *s*)]))#*xs*) ∈ *cptn-mod-nest-call*
using *c2incptn* **by** *auto*
then have (*c2*,*s*)#*xs*=(*map* (*lift c2*) [])@(*c2*, *snd*(*last* [(*Skip*, *s*)]))#*xs*
by (*auto simp add*:*lift-def*)

thus *?case* **using** *1 2 3* **by** (*simp add*: *CptnModNestSeq2*)
next
case (*SeqThrowc c2 s xs*)
have (n,Γ, [(*Throw*, *Normal s*)]) ∈ *cptn-mod-nest-call*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
then obtain *ys* **where**
ys-nil:*ys*=[] **and**
last:(n, Γ, (*Throw*, *Normal s*)#*ys*)∈ *cptn-mod-nest-call*
by *auto*
moreover have *fst* (*last* ((*Throw*, *Normal s*)#*ys*)) = *Throw* **using** *ys-nil last* **by** *auto*
moreover have *snd* (*last* ((*Throw*, *Normal s*)#*ys*)) = *Normal s* **using** *ys-nil last* **by** *auto*
moreover from *ys-nil* **have** (*map* (*lift c2*) *ys*) = [] **by** *auto*
ultimately show *?case* **using** *SeqThrowc.prems cptn-mod-nest-call.CptnModNestSeq3* **by** *fastforce*

next
case (*CondTruec s b c1 c2*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestCondT*)
next
case (*CondFalsec s b c1 c2*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestCondF*)
next
case (*WhileTruec s1 b c*)
have *sinb*: *s1*∈*b* **by** *fact*
have *SeqcWhile*: (n,Γ, (*Seq c* (*While b c*), *Normal s1*) # *xs*) ∈ *cptn-mod-nest-call*

by *fact*
have *divseq*:(∀ *s P Q zs*. (*Seq c* (*While b c*), *Normal s1*) # *xs*=(*Seq P Q*, *s*)#*zs* ⟶
(∃ *xs s'*. ((n,Γ,(P, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
(*zs*=(*map* (*lift Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (n,Γ,(Q, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift* (*Q*)) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*)))) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* ∧
(∃ *ys*. (n,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift Q*) *xs*)@((*Throw*,*Normal s'*)#*ys*))))))
)) **using** *div-seq-nest* [*OF SeqcWhile*] **by** (*auto simp add*: *seq-cond-nest-def*)
{fix *sa P Q zsa*
assume *ass*:(*Seq c* (*While b c*), *Normal s1*) # *xs* = (*Seq P Q*, *sa*) # *zsa*
then have *eqs*:*c* = *P* ∧ (*While b c*) = *Q* ∧ *Normal s1* = *sa* ∧ *xs* = *zsa* **by** *auto*
then have (∃ *xs s'*. (n,Γ, (*P*, *sa*) # *xs*) ∈ *cptn-mod-nest-call* ∧

```
                    (zsa = map (lift Q) xs ∨
                     fst (((P, sa) # xs) ! length xs) = Skip ∧
                         (∃ys. (n,Γ, (Q, snd (((P, sa) # xs) ! length xs)) # ys) ∈
cptn-mod-nest-call ∧
                          zsa = map (lift Q) xs @ (Q, snd (((P, sa) # xs) ! length
xs)) # ys) ∨
                    ((fst(((P, sa)#xs)!length xs)=Throw ∧
                      snd(last ((P, sa)#xs)) = Normal s′ ∧
                      (∃ys. (n,Γ,(Throw,Normal s′)#ys) ∈ cptn-mod-nest-call ∧
                  zsa=(map (lift Q) xs)@((Throw,Normal s′)#ys))
                    ))))
          using ass divseq by auto
   } note conc=this [of c While b c Normal s1 xs]
  then obtain xs′ s′
      where split:(n,Γ, (c, Normal s1) # xs′) ∈ cptn-mod-nest-call ∧
   (xs = map (lift (While b c)) xs′ ∨
    fst (((c, Normal s1) # xs′) ! length xs′) = Skip ∧
    (∃ys. (n,Γ, (While b c, snd (((c, Normal s1) # xs′) ! length xs′)) # ys)
         ∈ cptn-mod-nest-call ∧
         xs =
         map (lift (While b c)) xs′ @
         (While b c, snd (((c, Normal s1) # xs′) ! length xs′)) # ys) ∨
    fst (((c, Normal s1) # xs′) ! length xs′) = Throw ∧
    snd (last ((c, Normal s1) # xs′)) = Normal s′ ∧
    (∃ys. (n,Γ, ((Throw, Normal s′)#ys)) ∈ cptn-mod-nest-call ∧
    xs = map (lift (While b c)) xs′ @ ((Throw, Normal s′)#ys)))  by auto
 then have (xs = map (lift (While b c)) xs′ ∨
          fst (((c, Normal s1) # xs′) ! length xs′) = Skip ∧
          (∃ys. (n,Γ, (While b c, snd (((c, Normal s1) # xs′) ! length xs′)) # ys)
               ∈ cptn-mod-nest-call ∧
               xs =
               map (lift (While b c)) xs′ @
               (While b c, snd (((c, Normal s1) # xs′) ! length xs′)) # ys) ∨
          fst (((c, Normal s1) # xs′) ! length xs′) = Throw ∧
          snd (last ((c, Normal s1) # xs′)) = Normal s′ ∧
          (∃ys. (n,Γ, ((Throw, Normal s′)#ys)) ∈ cptn-mod-nest-call ∧
        xs = map (lift (While b c)) xs′ @ ((Throw, Normal s′)#ys))) ..
 thus ?case
 proof{
  assume 1:xs = map (lift (While b c)) xs′
  have 3:(n, Γ, (c, Normal s1) # xs′) ∈ cptn-mod-nest-call using split by auto

  then show ?thesis
    using 1 cptn-mod-nest-call.CptnModNestWhile1 sinb by fastforce
 next
  assume fst (((c, Normal s1) # xs′) ! length xs′) = Skip ∧
        (∃ys. (n,Γ, (While b c, snd (((c, Normal s1) # xs′) ! length xs′)) # ys)
             ∈ cptn-mod-nest-call ∧
             xs =
```

map (*lift* (*While b c*)) *xs'* @
(*While b c*, *snd* (((*c*, *Normal s1*) # *xs'*) ! *length xs'*)) # *ys*) ∨
fst (((*c*, *Normal s1*) # *xs'*) ! *length xs'*) = *Throw* ∧
snd (*last* ((*c*, *Normal s1*) # *xs'*)) = *Normal s'* ∧
(∃ *ys*. (*n*,Γ, ((*Throw*, *Normal s'*)#*ys*)) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift* (*While b c*)) *xs'* @ ((*Throw*, *Normal s'*)#*ys*))
thus *?case*
proof
assume *asm*:*fst* (((*c*, *Normal s1*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (*n*,Γ, (*While b c*, *snd* (((*c*, *Normal s1*) # *xs'*) ! *length xs'*)) # *ys*)
∈ *cptn-mod-nest-call* ∧
xs =
map (*lift* (*While b c*)) *xs'* @
(*While b c*, *snd* (((*c*, *Normal s1*) # *xs'*) ! *length xs'*)) # *ys*)
then obtain *ys*
where *asm'*:(*n*,Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs'*))) # *ys*)
∈ *cptn-mod-nest-call*
∧ *xs* = *map* (*lift* (*While b c*)) *xs'* @
(*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs'*))) # *ys*
by (*auto simp add*: *last-length*)
moreover have *3*:(*n*,Γ, (*c*, *Normal s1*) # *xs'*) ∈ *cptn-mod-nest-call* **using**
split **by** *auto*
moreover from *asm* **have** *fst* (*last* ((*c*, *Normal s1*) # *xs'*)) = *Skip*
by (*simp add*: *last-length*)
ultimately show *?case* **using** *sinb* **by** (*auto simp add*:*CptnModNestWhile2*)
next
assume *asm*: *fst* (((*c*, *Normal s1*) # *xs'*) ! *length xs'*) = *Throw* ∧
snd (*last* ((*c*, *Normal s1*) # *xs'*)) = *Normal s'* ∧
(∃ *ys*. (*n*,Γ, ((*Throw*, *Normal s'*)#*ys*)) ∈ *cptn-mod-nest-call* ∧
xs = *map* (*lift* (*While b c*)) *xs'* @ ((*Throw*, *Normal s'*)#*ys*))
moreover have *3*:(*n*,Γ, (*c*, *Normal s1*) # *xs'*) ∈ *cptn-mod-nest-call*
using *split* **by** *auto*
moreover from *asm* **have** *fst* (*last* ((*c*, *Normal s1*) # *xs'*)) = *Throw*
by (*simp add*: *last-length*)
ultimately show *?case* **using** *sinb* **by** (*auto simp add*:*CptnModNestWhile3*)
qed
}qed
next
case (*WhileFalsec s b c*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestSkip stepc.WhileFalsec*)
next
case (*Awaitc s b c t*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestSkip stepc.Awaitc*)
next
case (*AwaitAbruptc s b c t t'*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestThrow stepc.AwaitAbruptc*)

next
case (*Callc p bdy s*)

thus *?case* **using** *SmallStepCon.redex.simps*(*7*) **by** *auto*
next
case (*CallUndefinedc p s*)
then have $p = fn$ **by** *auto*
thus *?case* **using** *CallUndefinedc*
proof −
have (*LanguageCon.com.Call fn* $\cap_{gs}$ (*LanguageCon.com.Skip*::('b, 'a, 'c,'d) *LanguageCon.com*)) ≠ *Some LanguageCon.com.Skip*
by *simp*
then show *?thesis*
by (*metis* (*no-types*) *CallUndefinedc.hyps LanguageCon.com.inject*(*6*) *LanguageCon.inter-guards.simps*(*79*) *SmallStepCon.redex.simps*(*7*) ‹(*n*, Γ, (*LanguageCon.com.Skip*, *Stuck*) # *xs*) ∈ *cptn-mod-nest-call*› *cptn-mod-nest-call.CptnModNestSkip stepc.CallUndefinedc*)
qed
next
case (*DynComc c s*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestDynCom*)
next
case (*Catchc c1 s c1' s' c2*)
have *step*: $\Gamma\vdash_c$ (*c1*, *s*) → (*c1'*, *s'*) **by** (*simp add*: *Catchc.hyps*(*1*))
then have *nsc1*: *c1*≠*Skip* **using** *stepc-elim-cases*(*1*) **by** *blast*
have *assum*: (*n*,Γ, (*Catch c1' c2*, *s'*) # *xs*) ∈ *cptn-mod-nest-call*
using *Catchc.prems* **by** *blast*
have *divcatch*:(∀ *s P Q zs*. (*Catch c1' c2*, *s'*) # *xs*=(*Catch P Q*, *s*)#*zs* ⟶
(∃ *xs s' s''*. ((*n*,Γ,(*P*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
(*zs*=(*map* (*lift-catch Q*) *xs*) ∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *s*)#*xs*)) = *Normal s'* ∧ *s*=*Normal s''*∧
(∃ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call*
∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *s*)#*xs*)!*length xs*))#*ys*))))
∨
((*fst*(((*P*, *s*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call*
∧
zs=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *s*)#*xs*)))#*ys*))

))))
)) **using** *div-catch-nest* [*OF assum*] **by** (*auto simp add*: *catch-cond-nest-def*)
{**fix** *sa P Q zsa*
assume *ass*:(*Catch c1' c2*, *s'*) # *xs* = (*Catch P Q*, *sa*) # *zsa*
then have *eqs*:*c1'* = *P* ∧ *c2* = *Q* ∧ *s'* = *sa* ∧ *xs* = *zsa* **by** *auto*
then have (∃ *xs sv' sv''*. ((*n*, Γ,(*P*, *sa*)#*xs*) ∈ *cptn-mod-nest-call* ∧
(*zsa*=(*map* (*lift-catch Q*) *xs*) ∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Throw* ∧
snd(*last* ((*P*, *sa*)#*xs*)) = *Normal sv'* ∧ *s'*=*Normal sv''*∧
(∃ *ys*. (*n*,Γ,(*Q*, *snd*(((*P*, *sa*)#*xs*)!*length xs*))#*ys*) ∈ *cptn-mod-nest-call*
∧
zsa=(*map* (*lift-catch Q*) *xs*)@((*Q*, *snd*(((*P*, *sa*)#*xs*)!*length xs*))#*ys*))))

∨
((*fst*(((*P*, *sa*)#*xs*)!*length xs*)=*Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*P*, *sa*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call*
∧
zsa=(*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* ((*P*, *sa*)#*xs*)))#*ys*))))))
) **using** *ass divcatch* **by** *blast*
} **note** *conc*=*this* [*of c1′ c2 s′ xs*]
then obtain *xs′ sa′ sa′′*
where *split*:
(*n*,Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
(*xs* = *map* (*lift-catch c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Throw* ∧
snd (*last* ((*c1′*, *s′*) # *xs′*)) = *Normal sa′* ∧ *s′* = *Normal sa′′* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈ *cptn-mod-nest-call*
∧
xs = *map* (*lift-catch c2*) *xs′* @
(*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*) ∈ *cptn-mod-nest-call* ∧

xs=(*map* (*lift-catch c2*) *xs′*)@((*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*)))

by *blast*
then have (*xs* = *map* (*lift-catch c2*) *xs′* ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Throw* ∧
snd (*last* ((*c1′*, *s′*) # *xs′*)) = *Normal sa′* ∧ *s′* = *Normal sa′′* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∈ *cptn-mod-nest-call*
∧
xs = *map* (*lift-catch c2*) *xs′* @
(*c2*, *snd* (((*c1′*, *s′*) # *xs′*) ! *length xs′*)) # *ys*) ∨
fst (((*c1′*, *s′*) # *xs′*) ! *length xs′*) = *Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*) ∈ *cptn-mod-nest-call* ∧

xs=(*map* (*lift-catch c2*) *xs′*)@((*Skip*,*snd*(*last* ((*c1′*, *s′*)#*xs′*)))#*ys*)))

by *auto*
thus *?case*
proof{
assume *c1′nonf*:*xs* = *map* (*lift-catch c2*) *xs′*
then have *c1′cptn*:(*n*,Γ, (*c1′*, *s′*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *split* **by** *blast*
then have *induct-step*: (*n*, Γ, (*c1*, *s*) # (*c1′*, *s′*)#*xs′*) ∈ *cptn-mod-nest-call*
using *Catchc.hyps*(*2*) *Catchc.prems*(*2*) *SmallStepCon.redex.simps*(*11*) **by** *auto*
then have (*Catch c1′ c2*, *s′*)#*xs* = *map* (*lift-catch c2*) ((*c1′*, *s′*)#*xs′*)
using *c1′nonf*
by (*simp add*: *CptnModCatch1 lift-catch-def*)
thus *?thesis*
using *c1′nonf c1′cptn induct-step*

by (*auto simp add*: *CptnModNestCatch1*)
next
assume *fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Throw* ∧
snd (*last* ((*c1'*, *s'*) # *xs'*)) = *Normal sa'* ∧ *s'* = *Normal sa''* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs =*map* (*lift-catch c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∨
fst (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*c1'*, *s'*)#*xs'*)))#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift-catch c2*) *xs'*)@((*Skip*,*snd*(*last* ((*c1'*, *s'*)#*xs'*)))#*ys*))
thus *?thesis*
proof
assume *assth*:
fst (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Throw* ∧
snd (*last* ((*c1'*, *s'*) # *xs'*)) = *Normal sa'* ∧ *s'* = *Normal sa''* ∧
(∃ *ys*. (*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs =*map* (*lift-catch c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*)
then have *s'eqsa''*: *s'*=*Normal sa''* **by** *auto*
then have *snormal*: ∃ *ns*. *s*=*Normal ns* **by** (*metis Catchc.hyps*(*1*) *step-Abrupt-prop step-Fault-prop step-Stuck-prop xstate.exhaust*)
then obtain *ys*
where *split'*:(*n*,Γ, (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
xs =*map* (*lift-catch c2*) *xs'* @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*
using *assth* **by** *auto*
then have *c1'cptn*:(*n*,Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod-nest-call*
using *split* **by** *blast*
then have *induct-step*: (*n*,Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod-nest-call*
using *Catchc.hyps*(*2*) *Catchc.prems*(*2*) *SmallStepCon.redex.simps*(*11*) **by** *auto*
then have *seqmap*:(*Catch c1 c2*, *s*)#(*Catch c1' c2*, *s'*)#*xs* = *map* (*lift-catch c2*) ((*c1*,*s*)#(*c1'*, *s'*)#*xs'*) @ (*c2*, *snd* (((*c1'*, *s'*) # *xs'*) ! *length xs'*)) # *ys*
using *split'* **by** (*simp add*: *CptnModCatch3 lift-catch-def*)
then have *lastc1*:*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*) = ((*c1'*, *s'*) # *xs'*) ! *length xs'*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Throw*
using *assth* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Normal sa'*
using *assth* **by** *force*
thus *?thesis* **using** *snormal seqmap s'eqsa'' split'*
last-length cptn-mod-nest-call.CptnModNestCatch3
induct-step lastc1 lastc1skip
using *Cons-lift-catch-append* **by** *fastforce*

next
assume *assm*: *fst* (((*c1'*, *s'*) # *xs'*) ! *length xs'*) = *Skip* ∧
(∃ *ys*. (*n*,Γ,(*Skip*,*snd*(*last* ((*c1'*, *s'*)#*xs'*)))#*ys*) ∈ *cptn-mod-nest-call* ∧
xs=(*map* (*lift-catch c2*) *xs'*)@((*Skip*,*snd*(*last* ((*c1'*, *s'*)#*xs'*)))#*ys*))
then have *c1'cptn*:(*n*,Γ, (*c1'*, *s'*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *split* **by** *blast*
then have *induct-step*: (*n*,Γ, (*c1*, *s*) # (*c1'*, *s'*)#*xs'*) ∈ *cptn-mod-nest-call*
using *Catchc.hyps*(*2*) *Catchc.prems*(*2*) *SmallStepCon.redex.simps*(*11*) **by** *auto*
then have *map* (*lift-catch c2*) ((*c1'*, *s'*) # *xs'*) = (*Catch c1' c2*, *s'*) # *map* (*lift-catch c2*) *xs'*
by (*auto simp add*: *lift-catch-def*)
then obtain *ys*
where *seqmap*:(*Catch c1' c2*, *s'*)#*xs* = (*map* (*lift-catch c2*) ((*c1'*, *s'*)#*xs'*))@((*Skip*,*snd*(*last* ((*c1'*, *s'*)#*xs'*)))#*ys*)
using *assm* **by** *fastforce*
then have *lastc1*:*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*) = ((*c1'*, *s'*) # *xs'*) ! *length xs'*
by (*simp add*: *last-length*)
then have *lastc1skip*:*fst* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *Skip*
using *assm* **by** *fastforce*
then have *snd* (*last* ((*c1*, *s*) # (*c1'*, *s'*) # *xs'*)) = *snd* (*last* ((*c1'*, *s'*) # *xs'*))
using *assm* **by** *force*
thus *?thesis*
using *assm c1'cptn induct-step lastc1skip seqmap*
by (*auto simp add*:*cptn-mod-nest-call.CptnModNestCatch2*)
qed
}qed
next
case (*CatchThrowc c2 s*)
have *c2incptn*:(*n*,Γ, (*c2*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* **by** *fact*
then have *1*:(*n*,Γ, [(*Throw*, *Normal s*)]) ∈ *cptn-mod-nest-call*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
then have *2*:*fst*(*last* ([(*Throw*, *Normal s*)])) = *Throw* **by** *fastforce*
then have *3*:(*n*,Γ,(*c2*, *snd*(*last* [(*Throw*, *Normal s*)]))#*xs*) ∈ *cptn-mod-nest-call*

using *c2incptn* **by** *auto*
then have (*c2*,*Normal s*)#*xs*=(*map* (*lift c2*) [])@(*c2*, *snd*(*last* [(*Throw*, *Normal s*)]))#*xs*
by (*auto simp add*:*lift-def*)
thus *?case* **using** *1 2 3* **by** (*simp add*: *CptnModNestCatch3*)
next
case (*CatchSkipc c2 s*)
have (*n*,Γ, [(*Skip*, *s*)]) ∈ *cptn-mod-nest-call*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
then obtain *ys* **where**
ys-nil:*ys*=[] **and**

last:(n,Γ, (*Skip*, *s*)#*ys*)∈ *cptn-mod-nest-call*
by *auto*
moreover have *fst* (*last* ((*Skip*, *s*)#*ys*)) = *Skip* **using** *ys-nil last* **by** *auto*
moreover have *snd* (*last* ((*Skip*, *s*)#*ys*)) = *s* **using** *ys-nil last* **by** *auto*
moreover from *ys-nil* **have** (*map* (*lift-catch c2*) *ys*) = [] **by** *auto*
ultimately show *?case* **using** *CatchSkipc.prems*
by *simp* (*simp add*: *cptn-mod-nest-call.CptnModNestCatch2 ys-nil*)
next
case (*FaultPropc c f*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestSkip stepc.FaultPropc*)

next
case (*AbruptPropc c f*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestSkip stepc.AbruptPropc*)
next
case (*StuckPropc c*)
thus *?case* **by** (*simp add*: *cptn-mod-nest-call.CptnModNestSkip stepc.StuckPropc*)
qed

lemma *not-func-redex-cptn-mod-nest-n-env*:
assumes *a0*:$\Gamma\vdash_c$ (*P*,*s*) $\rightarrow_e$ (*P*, *t*) **and**
a1:(*n*,Γ,(*P*,*t*)#*xs*) ∈ *cptn-mod-nest-call*
shows (*n*,Γ,(*P*,*s*)#(*P*,*t*)#*xs*) ∈ *cptn-mod-nest-call*
by (*simp add*: *a0 a1 cptn-mod-nest-call.CptnModNestEnv*)

lemma *cptn-mod-nest-cptn-mod*:(*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call* ⟹ (Γ,*cfs*)∈ *cptn-mod*
by (*induct rule*:*cptn-mod-nest-call.induct*, (*fastforce simp*:*cptn-mod.intros*)+)

lemma *cptn-mod-cptn-mod-nest*: (Γ,*cfs*)∈ *cptn-mod* ⟹ ∃ *n*. (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call*
proof (*induct rule*:*cptn-mod.induct*)
case (*CptnModSkip* Γ *P s t xs*)
then obtain *n* **where** *cptn-nest*:(*n*, Γ, (*Skip*, *t*) # *xs*) ∈ *cptn-mod-nest-call* **by** *auto*
{**assume** *asm*:∀ *f*. ((∃ *sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Skip* ⟶ *P* ≠ *Call f*)
then have *?case* **using** *CptnModNestSkip*[*OF CptnModSkip*(*1*) *CptnModSkip*(*2*) *asm cptn-nest*] **by** *auto*
}**note** *t1*=*this*
{**assume** *asm*:¬ (∀ *f*. ((∃ *sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Skip* ⟶ *P* ≠ *Call f*))
then obtain *f* **where** *asm*:((∃ *sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Skip* ∧ *P* = *Call f*) **by** *auto*
then obtain *sn* **where** *normal-s*:*s*=*Normal sn* **by** *auto*
then have *t-eq-s*:*t*=*s* **using** *asm cptn-nest normal-s*

by (*metis CptnModSkip.hyps*(*1*) *LanguageCon.com.simps*(*22*)
LanguageCon.inter-guards.simps(*79*) *LanguageCon.inter-guards-Call*
Pair-inject stepc-Normal-elim-cases(*9*))
then have (*Suc n*, Γ,((*Call f*), *Normal sn*)#(*Skip*, *Normal sn*)#*xs*) ∈ *cptn-mod-nest-call*
using *asm cptn-nest normal-s CptnModNestCall* **by** *fastforce*
then have *?case* **using** *asm normal-s t-eq-s* **by** *fastforce*
}**note** *t2* = *this*
then show *?case* **using** *t1 t2* **by** *fastforce*
next
case (*CptnModThrow* Γ *P s t xs*)
then obtain *n* **where** *cptn-nest*:(*n*, Γ, (*Throw*, *t*) # *xs*) ∈ *cptn-mod-nest-call*
by *auto*
{**assume** *asm*:∀*f*. ((∃*sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Throw* ⟶ *P* ≠
Call f)
then have *?case* **using** *CptnModNestThrow*[*OF CptnModThrow*(*1*) *Cptn-ModThrow*(*2*) *asm cptn-nest*] **by** *auto*
}**note** *t1*=*this*
{**assume** *asm*:¬ (∀*f*. ((∃*sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Throw* ⟶ *P*
≠ *Call f*))
then obtain *f* **where** *asm*:((∃*sn*. *s* = *Normal sn*) ∧ (Γ *f*) = *Some Throw* ∧
P = *Call f*) **by** *auto*
then obtain *sn* **where** *normal-s*:*s*=*Normal sn* **by** *auto*
then have *t-eq-s*:*t*=*s* **using** *asm cptn-nest normal-s*
by (*metis CptnModThrow.hyps*(*1*) *LanguageCon.com.simps*(*22*)
LanguageCon.inter-guards.simps(*79*) *LanguageCon.inter-guards-Call*
Pair-inject stepc-Normal-elim-cases(*9*))
then have (*Suc n*, Γ,((*Call f*), *Normal sn*)#(*Throw*, *Normal sn*)#*xs*) ∈
cptn-mod-nest-call
using *asm cptn-nest normal-s CptnModNestCall* **by** *fastforce*
then have *?case* **using** *asm normal-s t-eq-s* **by** *fastforce*
}**note** *t2* = *this*
then show *?case* **using** *t1 t2* **by** *fastforce*
next
case (*CptnModSeq2* Γ *P0 s xs P1 ys zs*)
obtain *n* **where** *n*:(*n*, Γ, (*P0*, *s*) # *xs*) ∈ *cptn-mod-nest-call* **using** *CptnMod-Seq2*(*2*) **by** *auto*
also obtain *m* **where** *m*:(*m*, Γ, (*P1*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*) ∈
cptn-mod-nest-call
using *CptnModSeq2*(*5*) **by** *auto*
ultimately show *?case*
proof (*cases n*≥*m*)
case *True* **thus** *?thesis*
using *cptn-mod-nest-mono*[*of m* Γ - *n*] *m n CptnModSeq2 cptn-mod-nest-call.CptnModNestSeq2*
by *blast*
next
case *False*
thus *?thesis*
using *cptn-mod-nest-mono*[*of n* Γ - *m*] *m n CptnModSeq2*
cptn-mod-nest-call.CptnModNestSeq2 le-cases3 **by** *blast*

qed
next
case (*CptnModSeq3* Γ *P0 s xs s' ys zs P1*)
obtain *n* **where** *n*:(*n*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* **using** *CptnModSeq3*(*2*) **by** *auto*
also obtain *m* **where** *m*:(*m*, Γ, (*LanguageCon.com.Throw*, *Normal s'*) # *ys*) ∈ *cptn-mod-nest-call*
using *CptnModSeq3*(*6*) **by** *auto*
ultimately show *?case*
proof (*cases n≥m*)
case *True* **thus** *?thesis*
using *cptn-mod-nest-mono*[*of m* Γ *- n*] *m n CptnModSeq3 cptn-mod-nest-call.CptnModNestSeq3*
by *fastforce*
next
case *False*
thus *?thesis*
using *cptn-mod-nest-mono*[*of n* Γ *- m*] *m n CptnModSeq3*
cptn-mod-nest-call.CptnModNestSeq3 le-cases3
proof −
have *f1*: ¬ *n* ≤ *m* ∨ (*m*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
by (*metis cptn-mod-nest-mono*[*of n* Γ *- m*] *n*)
have *n* ≤ *m*
using *False* **by** *linarith*
then have (*m*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *f1* **by** *metis*
then show *?thesis*
by (*metis* (*no-types*) *CptnModSeq3*(*3*) *CptnModSeq3*(*4*) *CptnModSeq3*(*7*)

cptn-mod-nest-call.CptnModNestSeq3 m)
qed
qed
next
case (*CptnModWhile2* Γ *P s xs b zs ys*)
obtain *n* **where** *n*:(*n*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* **using** *CptnModWhile2*(*2*) **by** *auto*
also obtain *m* **where**
m: (*m*, Γ, (*LanguageCon.com.While b P*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈
cptn-mod-nest-call
using *CptnModWhile2*(*7*) **by** *auto*
ultimately show *?case*
proof (*cases n≥m*)
case *True* **thus** *?thesis*
using *cptn-mod-nest-mono*[*of m* Γ *- n*] *m n*
CptnModWhile2 cptn-mod-nest-call.CptnModNestWhile2 **by** *metis*
next
case *False*
thus *?thesis*
proof −

```
      have f1: ¬ n ≤ m ∨ (m, Γ, (P, Normal s) # xs) ∈ cptn-mod-nest-call
        using cptn-mod-nest-mono[of n Γ - m] n by presburger
      have n ≤ m
        using False by linarith
      then have (m, Γ, (P, Normal s) # xs) ∈ cptn-mod-nest-call
        using f1 by metis
      then show ?thesis
          by (metis (no-types) CptnModWhile2(3) CptnModWhile2(4) CptnMod-
While2(5)
              cptn-mod-nest-call.CptnModNestWhile2 m)
    qed
   qed
next
   case (CptnModWhile3 Γ P s xs b s' ys zs)
   obtain n where n:(n, Γ, (P, Normal s) # xs) ∈ cptn-mod-nest-call
     using CptnModWhile3(2) by auto
   also obtain m where
     m: (m, Γ, (LanguageCon.com.Throw, Normal s') # ys) ∈ cptn-mod-nest-call
     using CptnModWhile3(7) by auto
   ultimately show ?case
   proof (cases n≥m)
     case True thus ?thesis
     proof -
     have (n, Γ, (LanguageCon.com.Throw, Normal s') # ys) ∈ cptn-mod-nest-call
        using True cptn-mod-nest-mono[of m Γ - n] m by presburger
      then show ?thesis
        by (metis (no-types) CptnModWhile3.hyps(3) CptnModWhile3.hyps(4)
         CptnModWhile3.hyps(5) CptnModWhile3.hyps(8) cptn-mod-nest-call.CptnModNestWhile3
n)
     qed
   next
     case False
    thus ?thesis using m n cptn-mod-nest-call.CptnModNestWhile3 cptn-mod-nest-mono[of
n Γ - m]
       by (metis CptnModWhile3.hyps(3) CptnModWhile3.hyps(4)
           CptnModWhile3.hyps(5) CptnModWhile3.hyps(8) le-cases)
   qed
next
  case (CptnModCatch2 Γ P0 s xs ys zs P1)
  obtain n where n:(n, Γ, (P0, s) # xs) ∈ cptn-mod-nest-call using CptnMod-
Catch2(2) by auto
   also obtain m where m:(m, Γ, (LanguageCon.com.Skip, snd (last ((P0, s) #
xs))) # ys) ∈ cptn-mod-nest-call
     using CptnModCatch2(5) by auto
   ultimately show ?case
   proof (cases n≥m)
     case True thus ?thesis
       using cptn-mod-nest-mono[of m Γ - n] m n
            CptnModCatch2 cptn-mod-nest-call.CptnModNestCatch2 by blast
```

next
case *False*
thus *?thesis*
using *cptn-mod-nest-mono*[*of n* Γ - *m*] *m n CptnModCatch2*
cptn-mod-nest-call.CptnModNestCatch2 le-cases3 **by** *blast*
qed
next
case (*CptnModCatch3* Γ *P0 s xs s′ ys zs P1*)
obtain *n* **where** *n*:(*n*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *CptnModCatch3*(*2*) **by** *auto*
also obtain *m* **where** *m*:(*m*, Γ, (*ys*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *zs*) ∈ *cptn-mod-nest-call*
using *CptnModCatch3*(*6*) **by** *auto*
ultimately show *?case*
proof (*cases n*≥*m*)
case *True* **thus** *?thesis*
using *cptn-mod-nest-mono*[*of m* Γ - *n*] *m n CptnModCatch3 cptn-mod-nest-call.CptnModNestCatch3*
by *fastforce*
next
case *False*
thus *?thesis*
using *cptn-mod-nest-mono*[*of n* Γ - *m*] *m n CptnModCatch3*
cptn-mod-nest-call.CptnModNestCatch3 le-cases3
proof −
have *f1*: ¬ *n* ≤ *m* ∨ (*m*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using ‹⋀*cfs*. ⟦(*n*, Γ, *cfs*) ∈ *cptn-mod-nest-call*; *n* ≤ *m*⟧ ⟹ (*m*, Γ, *cfs*) ∈ *cptn-mod-nest-call*› *n* **by** *presburger*
have *n* ≤ *m*
using *False* **by** *auto*
then have (*m*, Γ, (*P0*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *f1* **by** *meson*
then show *?thesis*
by (*metis* (*no-types*) ‹*P1* = *map* (*lift-catch ys*) *xs* @ (*ys*, *snd* (*last* ((*P0*, *Normal s*) # *xs*))) # *zs*› ‹*fst* (*last* ((*P0*, *Normal s*) # *xs*)) = *LanguageCon.com.Throw*› ‹*snd* (*last* ((*P0*, *Normal s*) # *xs*)) = *Normal s′*› *cptn-mod-nest-call.CptnModNestCatch3* *m*)
qed
qed
qed(*fastforce intro*: *cptn-mod-nest-call.intros*)+

lemma *cptn-mod-eq-cptn-mod-nest*:
(Γ,*cfs*)∈ *cptn-mod* ⟷ (∃ *n*. (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call*)
using *cptn-mod-cptn-mod-nest cptn-mod-nest-cptn-mod* **by** *auto*

lemma *cptn-mod-eq-cptn-mod-nest′*:
∃ *n*. ((Γ,*cfs*)∈ *cptn-mod* ⟷ (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call*)
using *cptn-mod-eq-cptn-mod-nest* **by** *auto*

lemma *cptn-mod-eq-cptn-mod-nest1*:

(Γ,*cfs*)$\in$ *cptn-mod* = ($\exists$ *n*. (*n*,Γ,*cfs*) $\in$ *cptn-mod-nest-call*)
using *cptn-mod-cptn-mod-nest cptn-mod-nest-cptn-mod* **by** *auto*

lemma *cptn-eq-cptn-mod-nest*:
(Γ,*cfs*)$\in$ *cptn* = ($\exists$ *n*. (*n*,Γ,*cfs*) $\in$ *cptn-mod-nest-call*)
using *cptn-eq-cptn-mod-set cptn-mod-cptn-mod-nest cptn-mod-nest-cptn-mod* **by** *blast*

8.11 computation on nested calls limit

8.12 Elimination theorems

lemma *mod-env-not-component*:
shows $\neg$ $\Gamma\vdash_c$ (*P*, *s*) $\rightarrow$ (*P*, *t*)
proof
assume *a3*:$\Gamma\vdash_c$ (*P*, *s*) $\rightarrow$ (*P*, *t*)
thus *False* **using** *step-change-p-or-eq-s a3* **by** *fastforce*
qed

lemma *elim-cptn-mod-nest-step-c*:
assumes *a0*:(*n*,Γ,*cfg*) $\in$ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1*
shows $\Gamma\vdash_c$ (*P*,*s*) $\rightarrow$ (*Q*,*t*) $\vee$ $\Gamma\vdash_c$ (*P*,*s*) $\rightarrow_e$ (*Q*,*t*)
proof−
have (Γ,*cfg*) $\in$ *cptn* **using** *a0 cptn-mod-nest-cptn-mod*
using *cptn-eq-cptn-mod-set* **by** *auto*
then have $\Gamma\vdash_c$ (*P*,*s*) $\rightarrow_{ce}$ (*Q*,*t*) **using** *a1*
by (*metis c-step cptn-elim-cases*(*2*) *e-step*)
thus *?thesis*
using *step-ce-not-step-e-step-c* **by** *blast*
qed

lemma *elim-cptn-mod-nest-call-env*:
assumes *a0*:(*n*,Γ,*cfg*) $\in$ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*,*s*)#(*P*,*t*)#*cfg1* **and**
a2:$\forall$ *f*. Γ *f* = *Some* (*LanguageCon.com.Call f*) $\wedge$
($\exists$ *sn*. *s* = *Normal sn*) $\wedge$ *s* = *t* $\longrightarrow$ *SmallStepCon.redex P* $\neq$ *LanguageCon.com.Call f*
shows (*n*,Γ,(*P*,*t*)#*cfg1*) $\in$ *cptn-mod-nest-call*
using *a0 a1 a2*
proof (*induct arbitrary*: *P cfg1 s t rule*:*cptn-mod-nest-call.induct*)
case (*CptnModNestSeq1 n* Γ *P0 sa xs zs P1*)
then obtain *xs'* **where** *xs* = (*P0*, *t*)#*xs'* **unfolding** *lift-def* **by** *fastforce*
then have *step*:(*n*, Γ, (*P0*, *t*) # *xs'*) $\in$ *cptn-mod-nest-call* **using** *CptnModNestSeq1* **by** *fastforce*
have (*P*, *t*) = *lift P1* (*P0*, *t*) $\wedge$ *cfg1* = *map* (*lift P1*) *xs'*
using *CptnModNestSeq1.hyps*(*3*) *CptnModNestSeq1.prems*(*1*) ‹*xs* = (*P0*, *t*) # *xs'*› **by** *auto*
then have (*n*, Γ, (*LanguageCon.com.Seq P0 P1*, *t*) # *cfg1*) $\in$ *cptn-mod-nest-call*

by (*meson cptn-mod-nest-call.CptnModNestSeq1 local.step*)
then show *?case*
using *CptnModNestSeq1.prems*(*1*) **by** *fastforce*
next
case (*CptnModNestSeq2 n* Γ *P0 sa xs P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:*x*=(*P0*,*t*)
proof−
have *zs*=(*Seq P0 P1*,*t*)#*cfg1* **using** *Cons* **by** *fastforce*
thus *?thesis* **using** *Cons*(*7*) **unfolding** *lift-def*
proof −
assume *zs* = *map* (λ*a*. *case a of* (*P*, *s*) ⇒ (*LanguageCon.com.Seq P P1*, *s*)) (*x* # *xs'*) @
(*P1*, *snd* (*last* ((*P0*, *sa*) # *x* # *xs'*))) # *ys*
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0 P1* ∧ *snd x* = *t*
by (*simp add*: ‹*zs* = (*LanguageCon.com.Seq P0 P1*, *t*) # *cfg1*› *case-prod-beta*)
then show *?thesis*
by *fastforce*
qed
qed
then have *step*:(*n*, Γ, (*P0*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
have *fst* (*last* ((*P0*, *t*) # *xs'*)) = *LanguageCon.com.Skip*
using *Cons.prems*(*3*) ‹*x* = (*P0*, *t*)› **by** *force*
then show *?case*
using *Cons.prems*(*4*) *Cons.prems*(*6*) *CptnModNestSeq2.prems*(*1*) *x*
cptn-mod-nest-call.CptnModNestSeq2 local.step **by** *fastforce*
qed
next
case (*CptnModNestSeq3 n* Γ *P0 sa xs s' ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:*x*=(*P0*,*t*)
proof−
have *zs*:*zs*=(*Seq P0 P1*,*t*)#*cfg1* **using** *Cons* **by** *fastforce*
have (*LanguageCon.com.Seq* (*fst x*) *P1*, *snd x*) = *lift P1 x*
by (*simp add*: *lift-def prod.case-eq-if*)
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0 P1* ∧ *snd x* = *t*
using *Cons.prems*(*7*) *zs* **by** *force*
then show *?thesis*

by *fastforce*
qed
then have *step*:(*n*, Γ, (*P0*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
then obtain *t'* **where** *t*:*t*=*Normal t'*
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set x*
by (*metis snd-eqD*)
then show *?case* **using** *x Cons*(*5*) *Cons*(*6*) *cptn-mod-nest-call.CptnModNestSeq3 step*
proof −
have *last* ((*P0*, *Normal t'*) # *xs'*) = *last* ((*P0*, *Normal sa*) # *x* # *xs'*)
using *t x* **by** *force*
then have *fst* (*last* ((*P0*, *Normal t'*) # *xs'*)) = *LanguageCon.com.Throw*
using *Cons.prems*(*3*) **by** *presburger*
then show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)
CptnModNestSeq3.prems(*1*) *cptn-mod-nest-call.CptnModNestSeq3*
local.step t x **by** *fastforce*
qed
qed
next
case (*CptnModNestCatch1 n* Γ *P0 s xs zs P1*)
then obtain *xs'* **where** *xs* = (*P0*, *t*)#*xs'* **unfolding** *lift-catch-def* **by** *fastforce*
then have *step*:(*n*, Γ, (*P0*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *CptnModNest-Catch1* **by** *fastforce*
have (*P*, *t*) = *lift-catch P1* (*P0*, *t*) ∧ *cfg1* = *map* (*lift-catch P1*) *xs'*
using *CptnModNestCatch1.hyps*(*3*) *CptnModNestCatch1.prems*(*1*) ‹*xs* = (*P0*, *t*) # *xs'*› **by** *auto*
then have (*n*, Γ, (*Catch P0 P1*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
by (*meson cptn-mod-nest-call.CptnModNestCatch1 local.step*)
then show *?case*
using *CptnModNestCatch1.prems*(*1*) **by** *fastforce*
next
case (*CptnModNestCatch2 n* Γ *P0 sa xs ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:*x*=(*P0*,*t*)
proof−
have *zs*:*zs*=(*Catch P0 P1*,*t*)#*cfg1* **using** *Cons* **by** *fastforce*
have (*LanguageCon.com.Catch* (*fst x*) *P1*, *snd x*) = *lift-catch P1 x*
by (*simp add*: *lift-catch-def prod.case-eq-if*)
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch P0 P1* ∧ *snd x* = *t*
using *Cons.prems*(*6*) *zs* **by** *fastforce*
then show *?thesis*

by *fastforce*
qed
then have *step*:(n, Γ, (*P0*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
have *fst* (*last* ((*P0*, *t*) # *xs'*)) = *LanguageCon.com.Skip*
using *Cons.prems*(*3*) *x* **by** *auto*
then show *?case*
using *Cons.prems*(*4*) *Cons.prems*(*6*) *CptnModNestCatch2.prems*(*1*)
cptn-mod-nest-call.CptnModNestCatch2 local.step x **by** *fastforce*
qed
next
case (*CptnModNestCatch3 n* Γ *P0 sa xs s' P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:*x*=(*P0*,*t*)
proof−
have *zs*:*zs*=(*Catch P0 P1*,*t*)#*cfg1* **using** *Cons* **by** *fastforce*
thus *?thesis* **using** *Cons*(*8*) *lift-catch-def* **unfolding** *lift-def*
proof −
assume *zs* = *map* (*lift-catch P1*) (*x* # *xs'*) @ (*P1*, *snd* (*last* ((*P0*, *Normal sa*) # *x* # *xs'*))) # *ys*
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch P0 P1* ∧ *snd x* = *t*
by (*simp add*: *case-prod-unfold lift-catch-def zs*)
then show *?thesis*
by *fastforce*
qed
qed
then have *step*:(n, Γ, (*P0*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
then obtain t' **where** *t*:*t*=*Normal* t'
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set x*
by (*metis snd-eqD*)
then show *?case*
proof −
have *last* ((*P0*, *Normal t'*) # *xs'*) = *last* ((*P0*, *Normal sa*) # *x* # *xs'*)
using *t x* **by** *force*
then have *fst* (*last* ((*P0*, *Normal t'*) # *xs'*)) = *LanguageCon.com.Throw*
using *Cons.prems*(*3*) **by** *presburger*
then show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)
CptnModNestCatch3.prems(*1*) *cptn-mod-nest-call.CptnModNestCatch3*
local.step t x **by** *fastforce*
qed
qed

qed(*fastforce*+)

lemma *elim-cptn-mod-nest-not-env-call*:
assumes *a0*:(*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1* **and**
a2:(∀ *f*. *redex P* ≠ *Call f*) ∨
SmallStepCon.redex P = *LanguageCon.com.Call fn* ∧ Γ *fn* = *None* ∨
(*redex P* = *Call fn* ∧ (∀ *sa*. *s*≠*Normal sa*))
shows (*n*,Γ,(*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *a0 a1 a2*
proof (*induct arbitrary*: *P Q cfg1 s t rule*:*cptn-mod-nest-call.induct*)
case (*CptnModNestSeq1 n* Γ *P0 s xs zs P1*)
then obtain *P0′ xs′* **where** *xs* = (*P0′*, *t*)#*xs′* **unfolding** *lift-def* **by** *fastforce*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *CptnModNest-Seq1* **by** *fastforce*
have *Q*:(*Q*, *t*) = *lift P1* (*P0′*, *t*) ∧ *cfg1* = *map* (*lift P1*) *xs′*
using *CptnModNestSeq1.hyps*(*3*) *CptnModNestSeq1.prems*(*1*) ‹*xs* = (*P0′*, *t*) # *xs′*› **by** *auto*
also then have (*n*, Γ, (*LanguageCon.com.Seq P0′ P1*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
by (*meson cptn-mod-nest-call.CptnModNestSeq1 local.step*)
ultimately show *?case*
using *CptnModNestSeq1.prems*(*1*)
by (*simp add*: *Cons-lift Q*)
next
case (*CptnModNestSeq2 n* Γ *P0 sa xs P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs′*)
then have *x*:∃ *P0′*. *x*=(*P0′*,*t*)
proof−
obtain *P0″* **where** *zs*: *zs*=(*Seq P0″ P1*,*t*)#*cfg1* **using** *Cons*(*7*) *Cons*(*8*)
unfolding *lift-def* **by** (*simp add*: *Cons-eq-append-conv case-prod-beta′*)
thus *?thesis* **using** *Cons*(*7*) **unfolding** *lift-def*
proof −
assume *zs* = *map* (λ*a*. *case a of* (*P*, *s*) ⇒ (*LanguageCon.com.Seq P P1*, *s*)) (*x* # *xs′*) @
(*P1*, *snd* (*last* ((*P0*, *sa*) # *x* # *xs′*))) # *ys*
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0″ P1* ∧ *snd x* = *t*
by (*simp add*: *zs case-prod-beta*)
also have *sa*=*s* **using** *Cons* **by** *fastforce*
ultimately show *?thesis* **by** (*meson eq-snd-iff*)
qed
qed
then obtain *P0′* **where** *x*:*x*=(*P0′*,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *Cons* **by**

force
have *fst* (*last* ((*P0'*, *t*) # *xs'*)) = *LanguageCon.com.Skip*
using *Cons.prems*(*3*) *x* **by** *force*
then show *?case*
using *Cons.prems*(*4*) *Cons.prems*(*6*) *CptnModNestSeq2.prems*(*1*) *x*
local.step cptn-mod-nest-call.CptnModNestSeq2[*of n* Γ *P0' t xs' P1 ys*]
Cons-lift-append
by (*metis* (*no-types*, *lifting*) *last-ConsR list.inject list.simps*(*3*))
qed
next
case (*CptnModNestSeq3 n* Γ *P0 sa xs s' ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:∃ *P0'*. *x*=(*P0'*,*t*)
proof−
obtain *P0'* **where** *zs*:*zs*=(*Seq P0' P1*,*t*)#*cfg1* **using** *Cons*(*8*) *Cons*(*9*)
unfolding *lift-def*
unfolding *lift-def* **by** (*simp add*: *Cons-eq-append-conv case-prod-beta'*)
have (*LanguageCon.com.Seq* (*fst x*) *P1*, *snd x*) = *lift P1 x*
by (*simp add*: *lift-def prod.case-eq-if*)
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0'*
P1 ∧ *snd x* = *t*
using *zs* **by** (*simp add*: *Cons.prems*(*7*))
then show *?thesis* **by** (*meson eq-snd-iff*)
qed
then obtain *P0'* **where** *x*:*x*=(*P0'*,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0'*, *t*) # *xs'*) ∈ *cptn-mod-nest-call*
proof −
have *f1*: *LanguageCon.com.Seq P0 P1* = *P* ∧ *Normal sa* = *s*
using *CptnModNestSeq3.prems*(*1*) **by** *blast*
then have *SmallStepCon.redex P* = *SmallStepCon.redex P0*
by (*metis SmallStepCon.redex.simps*(*4*))
then show *?thesis*
using *f1 Cons.prems*(*2*) *CptnModNestSeq3.prems*(*2*) *x* **by** *presburger*
qed
then obtain *t'* **where** *t*:*t*=*Normal t'*
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set*
x
by (*metis snd-eqD*)
then show *?case* **using** *x Cons*(*5*) *Cons*(*6*) *cptn-mod-nest-call.CptnModNestSeq3*
step
proof −
have *last* ((*P0'*, *Normal t'*) # *xs'*) = *last* ((*P0*, *Normal sa*) # *x* # *xs'*)
using *t x* **by** *force*
also then have *fst* (*last* ((*P0'*, *Normal t'*) # *xs'*)) = *LanguageCon.com.Throw*
using *Cons.prems*(*3*) **by** *presburger*

ultimately show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)
CptnModNestSeq3.prems(*1*) *cptn-mod-nest-call.CptnModNestSeq3*[*of n* Γ *P0′ t′ xs′ s′ ys*]
local.step t x Cons-lift-append
by (*metis* (*no-types*, *lifting*) *list.sel*(*3*))
qed
qed
next
case (*CptnModNestCatch1 n* Γ *P0 s xs zs P1*)
then obtain *P0′ xs′* **where** *xs*:*xs* = (*P0′*, *t*)#*xs′* **unfolding** *lift-catch-def* **by** *fastforce*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *CptnModNest-Catch1* **by** *fastforce*
have *Q*:(*Q*, *t*) = *lift-catch P1* (*P0′*, *t*) ∧ *cfg1* = *map* (*lift-catch P1*) *xs′*
using *CptnModNestCatch1.hyps*(*3*) *CptnModNestCatch1.prems*(*1*) *xs* **by** *auto*
then have (*n*, Γ, (*Catch P0′ P1*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
by (*meson cptn-mod-nest-call.CptnModNestCatch1 local.step*)
then show *?case*
using *CptnModNestCatch1.prems*(*1*) **by** (*simp add*:*Cons-lift-catch Q*)
next
case (*CptnModNestCatch2 n* Γ *P0 sa xs ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs′*)
then have *x*:∃ *P0′*. *x*=(*P0′*,*t*)
proof−
obtain *P0′* **where** *zs*:*zs*=(*Catch P0′ P1*,*t*)#*cfg1* **using** *Cons* **unfolding** *lift-catch-def*
by (*simp add*: *case-prod-unfold*)
have (*LanguageCon.com.Catch* (*fst x*) *P1*, *snd x*) = *lift-catch P1 x*
by (*simp add*: *lift-catch-def prod.case-eq-if*)
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch P0′ P1* ∧ *snd x* = *t*
using *Cons.prems*(*6*) *zs* **by** *fastforce*
then show *?thesis* **by** (*meson eq-snd-iff*)
qed
then obtain *P0′* **where** *x*:*x*=(*P0′*,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call*
using *Cons.prems*(*2*) *CptnModNestCatch2.prems*(*1*) *CptnModNestCatch2.prems*(*2*) *x* **by** *force*

have *skip*:*fst* (*last* ((*P0′*, *t*) # *xs′*)) = *LanguageCon.com.Skip*
using *Cons.prems*(*3*) *x* **by** *auto*
show *?case*
proof −
have (*P*, *s*) # (*Q*, *t*) # *cfg1* = (*LanguageCon.com.Catch P0 P1*, *sa*) # *map*

(*lift-catch P1*) (*x* # *xs'*) @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *sa*) # *x* # *xs'*))) # *ys*
using *CptnModNestCatch2.prems* *Cons.prems*(*6*) **by** *auto*
then show *?thesis*
using *Cons-lift-catch-append Cons.prems*(*4*)
cptn-mod-nest-call.CptnModNestCatch2[*OF local.step skip*] *last.simps*
list.distinct(*1*)
x
by (*metis* (*no-types*) *list.sel*(*3*) *x*)
qed
qed
next
case (*CptnModNestCatch3 n* Γ *P0 sa xs s' P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs'*)
then have *x*:∃ *P0'*. *x*=(*P0'*,*t*)
proof−
obtain *P0'* **where** *zs*:*zs*=(*Catch P0' P1*,*t*)#*cfg1* **using** *Cons* **unfolding**
lift-catch-def
by (*simp add*: *case-prod-unfold*)
thus *?thesis* **using** *Cons*(*8*) *lift-catch-def* **unfolding** *lift-def*
proof −
assume *zs* = *map* (*lift-catch P1*) (*x* # *xs'*) @ (*P1*, *snd* (*last* ((*P0*, *Normal*
sa) # *x* # *xs'*))) # *ys*
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch*
P0' P1 ∧ *snd x* = *t*
by (*simp add*: *case-prod-unfold lift-catch-def zs*)
then show *?thesis* **by** (*meson eq-snd-iff*)
qed
qed
then obtain *P0'* **where** *x*:*x*=(*P0'*,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0'*, *t*) # *xs'*) ∈ *cptn-mod-nest-call* **using** *Cons*
using *Cons.prems*(*2*) *CptnModNestCatch3.prems*(*1*) *CptnModNestCatch3.prems*(*2*)
x **by** *force*
then obtain *t'* **where** *t*:*t*=*Normal t'*
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set*
x
by (*metis snd-eqD*)
then show *?case*
proof −
have *last* ((*P0'*, *Normal t'*) # *xs'*) = *last* ((*P0*, *Normal sa*) # *x* # *xs'*)
using *t x* **by** *force*
also then have *fst* (*last* ((*P0'*, *Normal t'*) # *xs'*)) = *LanguageCon.com.Throw*
using *Cons.prems*(*3*) **by** *presburger*
ultimately show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)

CptnModNestCatch3.prems(1) cptn-mod-nest-call.CptnModNestCatch3[of n Γ P0′ t′ xs′ s′ P1]
local.step t x **by** (*metis Cons-lift-catch-append list.sel(3)*)
qed
qed
next
case (*CptnModNestWhile1 n Γ P0 s′ xs b zs*)
thus *?case*
using *cptn-mod-nest-call.CptnModNestSeq1 list.inject* **by** *blast*
next
case (*CptnModNestWhile2 n Γ P0 s′ xs b zs ys*)
have (*LanguageCon.com.While b P0*, *Normal s′*) = (*P*, *s*) ∧
(*LanguageCon.com.Seq P0* (*LanguageCon.com.While b P0*), *Normal s′*) # *zs* = (*Q*, *t*) # *cfg1*
using *CptnModNestWhile2.prems* **by** *fastforce*
then show *?case*
using *CptnModNestWhile2.hyps(1) CptnModNestWhile2.hyps(3)*
CptnModNestWhile2.hyps(5) CptnModNestWhile2.hyps(6)
cptn-mod-nest-call.CptnModNestSeq2 **by** *blast*
next
case (*CptnModNestWhile3 n Γ P0 s′ xs b zs*) **thus** *?case*
by (*metis* (*no-types*) *CptnModNestWhile3.hyps(1) CptnModNestWhile3.hyps(3) CptnModNestWhile3.hyps(5)*
CptnModNestWhile3.hyps(6) CptnModNestWhile3.hyps(8) CptnModNestWhile3.prems
cptn-mod-nest-call.CptnModNestSeq3 list.inject)
qed(*fastforce+*)

inductive-cases *stepc-call-skip-normal*:
$\Gamma\vdash_c$(*Call p*,*Normal s*) → (*Skip*,*s′*)

lemma *elim-cptn-mod-nest-call-n-greater-zero*:
assumes *a0*:(*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*,*Normal s*)#(*Q*,*t*)#*cfg1* ∧ *P* = *Call f* ∧ Γ *f* = *Some Q* ∧ *P*≠*Q*
shows *n*>*0*
using *a0 a1* **by** (*induct rule*:*cptn-mod-nest-call.induct*, *fastforce+*)

lemma *elim-cptn-mod-nest-call-0-False*:
assumes *a0*:(*0*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*,*Normal s*)#(*Q*,*t*)#*cfg1* ∧ *P* = *Call f* ∧ Γ *f* = *Some Q* ∧ *P*≠*Q*
shows *PP*
using *a0 a1 elim-cptn-mod-nest-call-n-greater-zero*
by *fastforce*

lemma *elim-cptn-mod-nest-call-n-dec1*:
assumes *a0*:(*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**

a1:*cfg* = (*P*,*Normal s*)#(*Q*,*t*)#*cfg1* ∧ *P* = *Call f* ∧ Γ *f* = *Some Q* ∧ *t*= *Normal s* ∧ *P*≠*Q*
shows (*n*−*1*,Γ,(*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *a0 a1*
by (*induct rule*:*cptn-mod-nest-call.induct*,*fastforce*+)

lemma *elim-cptn-mod-nest-call-n-dec*:
assumes *a0*:(*n*,Γ,(*Call f*,*Normal s*)#(*the* (Γ *f*),*Normal s*)#*cfg1*) ∈ *cptn-mod-nest-call* **and**
a1:Γ *f* = *Some Q* **and** *a2*:*Call f*≠*the* (Γ *f*)
shows (*n*−*1*,Γ,(*the* (Γ *f*),*Normal s*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *elim-cptn-mod-nest-call-n-dec1*
using *a0 a1 a2*
by *fastforce*

lemma *elim-cptn-mod-nest-call-n*:
assumes *a0*:(*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a1:*cfg* = (*P*, *s*)#(*Q*,*t*)#*cfg1*
shows (*n*,Γ,(*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *a0 a1*
proof (*induct arbitrary*: *P Q cfg1 s t rule*:*cptn-mod-nest-call.induct*)
case (*CptnModNestCall n* Γ *bdy sa ys p*)
thus *?case* **using** *cptn-mod-nest-mono1 list.inject* **by** *blast*
next
case (*CptnModNestSeq1 n* Γ *P0 s xs zs P1*)
then obtain *P0′ xs′* **where** *xs* = (*P0′*, *t*)#*xs′* **unfolding** *lift-def* **by** *fastforce*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *CptnModNestSeq1* **by** *fastforce*
have *Q*:(*Q*, *t*) = *lift P1* (*P0′*, *t*) ∧ *cfg1* = *map* (*lift P1*) *xs′*
using *CptnModNestSeq1.hyps*(*3*) *CptnModNestSeq1.prems*(*1*) ‹*xs* = (*P0′*, *t*) # *xs′*› **by** *auto*
also then have (*n*, Γ, (*LanguageCon.com.Seq P0′ P1*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
by (*meson cptn-mod-nest-call.CptnModNestSeq1 local.step*)
ultimately show *?case*
using *CptnModNestSeq1.prems*(*1*)
by (*simp add*: *Cons-lift Q*)
next
case (*CptnModNestSeq2 n* Γ *P0 sa xs P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs′*)
then have *x*:∃ *P0′*. *x*=(*P0′*,*t*)
proof−
obtain *P0″* **where** *zs*: *zs*=(*Seq P0″ P1*,*t*)#*cfg1* **using** *Cons*(*7*) *Cons*(*8*)
unfolding *lift-def* **by** (*simp add*: *Cons-eq-append-conv case-prod-beta′*)
thus *?thesis* **using** *Cons*(*7*) **unfolding** *lift-def*

proof −
assume *zs* = *map* (λ*a*. *case a of* (*P*, *s*) ⇒ (*LanguageCon.com.Seq P P1*, *s*)) (*x* # *xs*′) @
(*P1*, *snd* (*last* ((*P0*, *sa*) # *x* # *xs*′))) # *ys*
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0*′′ *P1* ∧ *snd x* = *t*
by (*simp add*: *zs case-prod-beta*)
also have *sa*=*s* **using** *Cons* **by** *fastforce*
ultimately show *?thesis* **by** (*meson eq-snd-iff*)
qed
qed
then obtain *P0*′ **where** *x*:*x*=(*P0*′,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0*′, *t*) # *xs*′) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *force*
have *fst* (*last* ((*P0*′, *t*) # *xs*′)) = *LanguageCon.com.Skip*
using *Cons.prems*(*3*) *x* **by** *force*
then show *?case*
using *Cons.prems*(*4*) *Cons.prems*(*6*) *CptnModNestSeq2.prems*(*1*) *x*
local.step cptn-mod-nest-call.CptnModNestSeq2[*of n* Γ *P0*′ *t xs*′ *P1 ys*] *Cons-lift-append*
by (*metis* (*no-types*, *lifting*) *last-ConsR list.inject list.simps*(*3*))
qed
next
case (*CptnModNestSeq3 n* Γ *P0 sa xs s*′ *ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs*′)
then have *x*:∃ *P0*′. *x*=(*P0*′,*t*)
proof−
obtain *P0*′ **where** *zs*:*zs*=(*Seq P0*′ *P1*,*t*)#*cfg1* **using** *Cons*(*8*) *Cons*(*9*)
unfolding *lift-def*
unfolding *lift-def* **by** (*simp add*: *Cons-eq-append-conv case-prod-beta*′)
have (*LanguageCon.com.Seq* (*fst x*) *P1*, *snd x*) = *lift P1 x*
by (*simp add*: *lift-def prod.case-eq-if*)
then have *LanguageCon.com.Seq* (*fst x*) *P1* = *LanguageCon.com.Seq P0*′ *P1* ∧ *snd x* = *t*
using *zs* **by** (*simp add*: *Cons.prems*(*7*))
then show *?thesis* **by** (*meson eq-snd-iff*)
qed
then obtain *P0*′ **where** *x*:*x*=(*P0*′,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0*′, *t*) # *xs*′) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
then obtain *t*′ **where** *t*:*t*=*Normal t*′
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set x*
by (*metis snd-eqD*)
then show *?case* **using** *x Cons*(*5*) *Cons*(*6*) *cptn-mod-nest-call.CptnModNestSeq3*

step
proof –
have *last* $((P0', Normal\ t') \# xs') = last\ ((P0, Normal\ sa) \# x \# xs')$
using *t x* by *force*
also then have *fst* $(last\ ((P0', Normal\ t') \# xs')) = LanguageCon.com.Throw$
using *Cons.prems*(*3*) by *presburger*
ultimately show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)
CptnModNestSeq3.prems(*1*) *cptn-mod-nest-call.CptnModNestSeq3*[*of n* Γ *P0′ t′ xs′ s′ ys*]
local.step t x *Cons-lift-append*
by (*metis* (*no-types*, *lifting*) *list.sel*(*3*))
qed
qed
next
case (*CptnModNestCatch1 n* Γ *P0 s xs zs P1*)
then obtain *P0′ xs′* where *xs*:*xs* = (*P0′*, *t*)#*xs′* unfolding *lift-catch-def* by *fastforce*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* using *CptnModNest-Catch1* by *fastforce*
have *Q*:(*Q*, *t*) = *lift-catch P1* (*P0′*, *t*) ∧ *cfg1* = *map* (*lift-catch P1*) *xs′*
using *CptnModNestCatch1.hyps*(*3*) *CptnModNestCatch1.prems*(*1*) *xs* by *auto*
then have (*n*, Γ, (*Catch P0′ P1*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
by (*meson cptn-mod-nest-call.CptnModNestCatch1 local.step*)
then show *?case*
using *CptnModNestCatch1.prems*(*1*) by (*simp add*:*Cons-lift-catch Q*)
next
case (*CptnModNestCatch2 n* Γ *P0 sa xs ys zs P1*)
thus *?case*
proof (*induct xs*)
case *Nil* thus *?case* using *Nil.prems*(*6*) *Nil.prems*(*7*) by *force*
next
case (*Cons x xs′*)
then have *x*:∃ *P0′*. *x*=(*P0′*,*t*)
proof–
obtain *P0′* where *zs*:*zs*=(*Catch P0′ P1*,*t*)#*cfg1* using *Cons* unfolding *lift-catch-def*
by (*simp add*: *case-prod-unfold*)
have (*LanguageCon.com.Catch* (*fst x*) *P1*, *snd x*) = *lift-catch P1 x*
by (*simp add*: *lift-catch-def prod.case-eq-if*)
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch P0′ P1* ∧ *snd x* = *t*
using *Cons.prems*(*6*) *zs* by *fastforce*
then show *?thesis* by (*meson eq-snd-iff*)
qed
then obtain *P0′* where *x*:*x*=(*P0′*,*t*) by *auto*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* using *Cons* by *fastforce*
have *skip*:*fst* (*last* ((*P0′*, *t*) # *xs′*)) = *LanguageCon.com.Skip*

using *Cons.prems*(*3*) *x* **by** *auto*
show *?case*
proof −
have (*P*, *s*) # (*Q*, *t*) # *cfg1* = (*LanguageCon.com.Catch P0 P1*, *sa*) # *map* (*lift-catch P1*) (*x* # *xs′*) @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *sa*) # *x* # *xs′*))) # *ys*
using *CptnModNestCatch2.prems Cons.prems*(*6*) **by** *auto*
then show *?thesis*
using *Cons-lift-catch-append Cons.prems*(*4*)
cptn-mod-nest-call.CptnModNestCatch2[*OF local.step skip*] *last.simps list.distinct*(*1*)
x
by (*metis* (*no-types*) *list.sel*(*3*) *x*)
qed
qed
next
case (*CptnModNestCatch3 n* Γ *P0 sa xs s′ P1 ys zs*)
thus *?case*
proof (*induct xs*)
case *Nil* **thus** *?case* **using** *Nil.prems*(*6*) *Nil.prems*(*7*) **by** *force*
next
case (*Cons x xs′*)
then have *x*:∃ *P0′*. *x*=(*P0′*,*t*)
proof−
obtain *P0′* **where** *zs*:*zs*=(*Catch P0′ P1*,*t*)#*cfg1* **using** *Cons* **unfolding** *lift-catch-def*
by (*simp add*: *case-prod-unfold*)
thus *?thesis* **using** *Cons*(*8*) *lift-catch-def* **unfolding** *lift-def*
proof −
assume *zs* = *map* (*lift-catch P1*) (*x* # *xs′*) @ (*P1*, *snd* (*last* ((*P0*, *Normal sa*) # *x* # *xs′*))) # *ys*
then have *LanguageCon.com.Catch* (*fst x*) *P1* = *LanguageCon.com.Catch P0′ P1* ∧ *snd x* = *t*
by (*simp add*: *case-prod-unfold lift-catch-def zs*)
then show *?thesis* **by** (*meson eq-snd-iff*)
qed
qed
then obtain *P0′* **where** *x*:*x*=(*P0′*,*t*) **by** *auto*
then have *step*:(*n*, Γ, (*P0′*, *t*) # *xs′*) ∈ *cptn-mod-nest-call* **using** *Cons* **by** *fastforce*
then obtain *t′* **where** *t*:*t*=*Normal t′*
using *Normal-Normal Cons*(*2*) *Cons*(*5*) *cptn-mod-nest-cptn-mod cptn-eq-cptn-mod-set x*
by (*metis snd-eqD*)
then show *?case*
proof −
have *last* ((*P0′*, *Normal t′*) # *xs′*) = *last* ((*P0*, *Normal sa*) # *x* # *xs′*)
using *t x* **by** *force*
also then have *fst* (*last* ((*P0′*, *Normal t′*) # *xs′*)) = *LanguageCon.com.Throw*

using *Cons.prems*(*3*) **by** *presburger*
ultimately show *?thesis*
using *Cons.prems*(*4*) *Cons.prems*(*5*) *Cons.prems*(*7*)
CptnModNestCatch3.prems(*1*) *cptn-mod-nest-call.CptnModNestCatch3*[*of*
n Γ *P0′ t′ xs′ s′ P1*]
local.step t x **by** (*metis Cons-lift-catch-append list.sel*(*3*))
qed
qed
next
case (*CptnModNestWhile1 n* Γ *P0 s′ xs b zs*)
thus *?case*
using *cptn-mod-nest-call.CptnModNestSeq1 list.inject* **by** *blast*
next
case (*CptnModNestWhile2 n* Γ *P0 s′ xs b zs ys*)
have (*LanguageCon.com.While b P0*, *Normal s′*) = (*P*, *s*) ∧
(*LanguageCon.com.Seq P0* (*LanguageCon.com.While b P0*), *Normal s′*) #
zs = (*Q*, *t*) # *cfg1*
using *CptnModNestWhile2.prems* **by** *fastforce*
then show *?case*
using *CptnModNestWhile2.hyps*(*1*) *CptnModNestWhile2.hyps*(*3*)
CptnModNestWhile2.hyps(*5*) *CptnModNestWhile2.hyps*(*6*)
cptn-mod-nest-call.CptnModNestSeq2 **by** *blast*
next
case (*CptnModNestWhile3 n* Γ *P0 s′ xs b zs*) **thus** *?case*
by (*metis* (*no-types*) *CptnModNestWhile3.hyps*(*1*) *CptnModNestWhile3.hyps*(*3*)
CptnModNestWhile3.hyps(*5*)
CptnModNestWhile3.hyps(*6*) *CptnModNestWhile3.hyps*(*8*)
CptnModNestWhile3.prems
cptn-mod-nest-call.CptnModNestSeq3 list.inject)
qed (*fastforce*+)

definition *min-call* **where**
min-call n Γ *cfs* ≡ (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call* ∧ (∀ *m*<*n*. ¬((*m*,Γ,*cfs*) ∈
cptn-mod-nest-call))

lemma *minimum-nest-call*:
(*m*,Γ,*cfs*) ∈ *cptn-mod-nest-call* ⟹
∃ *n*. *min-call n* Γ *cfs*
unfolding *min-call-def*
proof (*induct arbitrary*: *m rule*:*cptn-mod-nest-call.induct*)
case (*CptnModNestOne*) **thus** *?case* **using** *cptn-mod-nest-call.CptnModNestOne*
by *blast*
next
case (*CptnModNestEnv* Γ *P s t n xs*)
then have ¬ $\Gamma\vdash_c$ (*P*, *s*) → (*P*, *t*)
using *mod-env-not-component step-change-p-or-eq-s* **by** *blast*

then obtain *min-n* **where** *min*:(*min-n*, Γ, (*P*, *t*) # *xs*) ∈ *cptn-mod-nest-call* ∧

(∀ *m*<*min-n*. (*m*, Γ, (*P*, *t*) # *xs*) ∉ *cptn-mod-nest-call*)
using *CptnModNestEnv* **by** *blast*
then have (*min-n*, Γ, (*P*,*s*)#(*P*, *t*) # *xs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-call.CptnModNestEnv CptnModNestEnv* **by** *blast*
also have (∀ *m*<*min-n*. (*m*, Γ, (*P*, *s*)#(*P*, *t*) # *xs*) ∉ *cptn-mod-nest-call*)
using *elim-cptn-mod-nest-call-n min* **by** *fastforce*
ultimately show *?case* **by** *auto*
next
case (*CptnModNestSkip* Γ *P s t n xs*)
then obtain *min-n* **where**
min:(*min-n*, Γ, (*LanguageCon.com.Skip*, *t*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-n*. (*m*, Γ, (*LanguageCon.com.Skip*, *t*) # *xs*) ∉ *cptn-mod-nest-call*)

by *auto*
then have (*min-n*, Γ, (*P*,*s*)#(*LanguageCon.com.Skip*, *t*) # *xs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-call.CptnModNestSkip CptnModNestSkip* **by** *blast*
also have (∀ *m*<*min-n*. (*m*, Γ, (*P*, *s*)#(*LanguageCon.com.Skip*, *t*) # *xs*) ∉ *cptn-mod-nest-call*)
using *elim-cptn-mod-nest-call-n min* **by** *blast*
ultimately show *?case* **by** *fastforce*
next
case (*CptnModNestThrow* Γ *P s t n xs*) **thus** *?case*
by (*meson cptn-mod-nest-call.CptnModNestThrow elim-cptn-mod-nest-call-n*)

next
case (*CptnModNestCondT n* Γ *P0 s xs b P1*) **thus** *?case*
by (*meson cptn-mod-nest-call.CptnModNestCondT elim-cptn-mod-nest-call-n*)
next
case (*CptnModNestCondF n* Γ *P1 s xs b P0*) **thus** *?case*
by (*meson cptn-mod-nest-call.CptnModNestCondF elim-cptn-mod-nest-call-n*)
next
case (*CptnModNestSeq1 n* Γ *P s xs zs Q*) **thus** *?case*
by (*metis* (*no-types*, *lifting*) *Seq-P-Not-finish cptn-mod-nest-call.CptnModNestSeq1 div-seq-nest*)
next
case (*CptnModNestSeq2 n* Γ *P s xs Q ys zs*)
then obtain *min-p* **where**
min-p:(*min-p*, Γ, (*P*, *s*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-p*. (*m*, Γ, (*P*, *s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestSeq2*(*5*) **obtain** *min-q* **where**
min-q:(*min-q*, Γ, (*Q*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-q*. (*m*, Γ, (*Q*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*) ∉ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof(*cases min-p*≥*min-q*)
case *True*

then have (*min-p*, Γ, (*Q*, *snd* (*last* ((*P*,*s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-q* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-p*, Γ, (*Seq P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-p*] *cptn-mod-nest-call.CptnModNestSeq2*[*of min-p* Γ *P s xs Q ys zs*]
CptnModNestSeq2(*6*) *CptnModNestSeq2*(*3*)
by *blast*
also have ∀ *m*<*min-p*. (*m*, Γ,(*Seq P Q*,*s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestSeq2.hyps*(*3*) *CptnModNestSeq2.hyps*(*6*) *Seq-P-Ends-Normal div-seq-nest min-p*)
ultimately show *?thesis* **by** *auto*
next
case *False*
then have (*min-q*, Γ, (*P*, *s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have (*min-q*, Γ, (*Seq P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-q*] *cptn-mod-nest-call.CptnModNestSeq2*[*of min-q* Γ *P s xs Q ys zs*]
CptnModNestSeq2(*6*) *CptnModNestSeq2*(*3*)
by *blast*
also have ∀ *m*<*min-q*. (*m*, Γ,(*Seq P Q*,*s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**fix** *m*
assume *min-m*:*m*<*min-q*
then have (*m*, Γ,(*Seq P Q*, *s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**assume** *ass*:(*m*, Γ, (*Seq P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
then obtain *xs′ s′ s″* **where**
m-cptn:(*m*, Γ, (*P*, *s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
seq-cond-nest zs Q xs′ P s s″ s′ Γ *m*
using
div-seq-nest[*of m* Γ (*LanguageCon.com.Seq P Q*, *s*) # *zs*]
by *fastforce*
then have *seq-cond-nest zs Q xs′ P s s″ s′* Γ *m* **by** *auto*
then have *?thesis*
using *Seq-P-Ends-Normal*[*OF CptnModNestSeq2*(*6*) *CptnModNestSeq2*(*3*) *ass*]
min-m min-q
by (*metis last-length*)
} **thus** *?thesis* **by** *auto*
qed
}**thus** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*CptnModNestSeq3 n* Γ *P s xs s′ ys zs Q*)
then obtain *min-p* **where**
min-p:(*min-p*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧

($\forall$ *m*<*min-p*. (*m*, Γ, (*P*, *Normal s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestSeq3*(*6*) **obtain** *min-q* **where**
min-q:(*min-q*, Γ, (*Throw*, *Normal s'*) # *ys*) ∈ *cptn-mod-nest-call* ∧
($\forall$ *m*<*min-q*. (*m*, Γ, (*Throw*, *Normal s'*) # *ys*) ∉ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof(*cases min-p*≥*min-q*)
case *True*
then have (*min-p*, Γ, (*Throw*, *Normal s'*) # *ys*) ∈ *cptn-mod-nest-call*
using *min-q* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-p*, Γ, (*Seq P Q*, *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-p*] *cptn-mod-nest-call*.*CptnModNestSeq3*[*of min-p* Γ *P s xs s' ys zs Q*]
CptnModNestSeq3(*4*) *CptnModNestSeq3*(*3*) *CptnModNestSeq3*(*7*)
by *blast*
also have $\forall$ *m*<*min-p*. (*m*, Γ,(*Seq P Q*,*Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestSeq3*.*hyps*(*3*) *CptnModNestSeq3*.*hyps*(*4*) *CptnModNestSeq3*.*hyps*(*7*) *Seq-P-Ends-Abort div-seq-nest min-p*)
ultimately show *?thesis* **by** *auto*
next
case *False*
then have (*min-q*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have (*min-q*, Γ, (*Seq P Q*, *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-q*] *cptn-mod-nest-call*.*CptnModNestSeq3*[*of min-q* Γ *P s xs s' ys zs Q*]
CptnModNestSeq3(*4*) *CptnModNestSeq3*(*3*) *CptnModNestSeq3*(*7*)
by *blast*
also have $\forall$ *m*<*min-q*. (*m*, Γ,(*Seq P Q*,*Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestSeq3*.*hyps*(*3*) *CptnModNestSeq3*.*hyps*(*4*) *CptnModNestSeq3*.*hyps*(*7*) *Seq-P-Ends-Abort div-seq-nest min-q*)
ultimately show *?thesis* **by** *auto*
qed
next
case (*CptnModNestWhile1 n* Γ *P s xs b zs*)
then obtain *min-n* **where**
min:(*min-n*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧
($\forall$ *m*<*min-n*. (*m*, Γ, (*P*, *Normal s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
then have (*min-n*, Γ, (*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-call*.*CptnModNestWhile1*[*of min-n* Γ *P s xs b zs*] *CptnModNestWhile1*
by *meson*
also have $\forall$ *m*<*min-n*. (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestWhile1*.*hyps*(*4*) *Seq-P-Not-finish div-seq-nest elim-cptn-mod-nest-call-n min*)
ultimately show *?case* **by** *auto*

next
case (*CptnModNestWhile2 n* Γ *P s xs b zs ys*)
then obtain *min-n-p* **where**
min-p:(*min-n-p*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-n-p*. (*m*, Γ, (*P*, *Normal s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestWhile2* **obtain** *min-n-w* **where**
min-w:(*min-n-w*, Γ, (*LanguageCon.com.While b P*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-n-w*. (*m*, Γ, (*LanguageCon.com.While b P*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*)
∉ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof (*cases min-n-p*≥*min-n-w*)
case *True*
then have (*min-n-p*, Γ,
(*LanguageCon.com.While b P*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-w* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-n-p*, Γ, (*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-call.CptnModNestWhile2*[*of min-n-p* Γ *P s xs b zs*] *CptnModNestWhile2*
by *blast*
also have ∀ *m*<*min-n-p*. (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestWhile2.hyps*(*3*) *CptnModNestWhile2.hyps*(*5*)
Seq-P-Ends-Normal div-seq-nest elim-cptn-mod-nest-call-n min-p)
ultimately show *?thesis* **by** *auto*
next
case *False*
then have *False*:*min-n-p*<*min-n-w* **by** *auto*
then have (*min-n-w*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have (*min-n-w*, Γ, (*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *min-w min-p cptn-mod-nest-call.CptnModNestWhile2*[*of min-n-w* Γ *P s xs b zs*] *CptnModNestWhile2*
by *blast*
also have ∀ *m*<*min-n-w*. (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**fix** *m*
assume *min-m*:*m*<*min-n-w*
then have (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**assume** (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) #

zs) ∈ *cptn-mod-nest-call*
then have *a1*:(*m*, Γ,(*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*

using *elim-cptn-mod-nest-not-env-call* **by** *fastforce*
then obtain *xs′ s′ s″* **where**
m-cptn:(*m*, Γ, (*P*, *Normal s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
seq-cond-nest zs (*While b P*) *xs′ P* (*Normal s*) *s″ s′* Γ *m*
using
div-seq-nest[*of m* Γ (*LanguageCon.com.Seq P* (*LanguageCon.com.While b P*), *Normal s*) # *zs*]
by *fastforce*
then have *seq-cond-nest zs* (*While b P*) *xs′ P* (*Normal s*) *s″ s′* Γ *m* **by** *auto*
then have *?thesis* **unfolding** *seq-cond-nest-def*
by (*metis CptnModNestWhile2.hyps*(*3*) *CptnModNestWhile2.hyps*(*5*) *Seq-P-Ends-Normal a1 last-length m-cptn min-m min-w*)
} thus *?thesis* **by** *auto*
qed
}thus *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*CptnModNestWhile3 n* Γ *P s xs b s′ ys zs*)
then obtain *min-n-p* **where**
min-p:(*min-n-p*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-n-p*. (*m*, Γ, (*P*, *Normal s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestWhile3* **obtain** *min-n-w* **where**
min-w:(*min-n-w*, Γ, (*Throw*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-n-w*. (*m*, Γ, (*Throw*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∉ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof (*cases min-n-p*≥*min-n-w*)
case *True*
then have (*min-n-p*, Γ,
(*Throw*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-w* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-n-p*, Γ, (*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-call.CptnModNestWhile3*[*of min-n-p* Γ *P s xs b s′ ys zs*]
CptnModNestWhile3
by *fastforce*
also have ∀ *m*<*min-n-p*. (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
by (*metis CptnModNestWhile3.hyps*(*3*) *CptnModNestWhile3.hyps*(*5*) *CptnModNestWhile3.hyps*(*8*)

Seq-P-Ends-Abort div-seq-nest elim-cptn-mod-nest-call-n min-p)
ultimately show *?thesis* **by** *auto*
next
case *False*
then have *False*:*min-n-p*<*min-n-w* **by** *auto*
then have (*min-n-w*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have (*min-n-w*, Γ, (*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *min-w min-p cptn-mod-nest-call.CptnModNestWhile3*[*of min-n-w* Γ *P s xs b s′ ys zs*]
CptnModNestWhile3
by *fastforce*
also have ∀ *m*<*min-n-w*. (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**fix** *m*
assume *min-m*:*m*<*min-n-w*
then have (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**assume** (*m*, Γ,(*While b P*, *Normal s*) # (*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
then have *s1*:(*m*, Γ,(*Seq P* (*While b P*), *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*

using *elim-cptn-mod-nest-not-env-call* **by** *fastforce*
then obtain *xs′ s′ s″* **where**
m-cptn:(*m*, Γ, (*P*, *Normal s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
seq-cond-nest zs (*While b P*) *xs′ P* (*Normal s*) *s″ s′* Γ *m*
using
div-seq-nest[*of m* Γ (*LanguageCon.com.Seq P* (*LanguageCon.com.While b P*), *Normal s*) # *zs*]
by *fastforce*
then have *seq-cond-nest zs* (*While b P*) *xs′ P* (*Normal s*) *s″ s′* Γ *m* **by** *auto*
then have *?thesis* **unfolding** *seq-cond-nest-def*
by (*metis CptnModNestWhile3.hyps*(*3*) *CptnModNestWhile3.hyps*(*5*) *CptnModNestWhile3.hyps*(*8*) *Seq-P-Ends-Abort s1 m-cptn min-m min-w*)
} **thus** *?thesis* **by** *auto*
qed
}**thus** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*CptnModNestCall n* Γ *bdy s xs f*) **thus** *?case*
proof −
{ **fix** *nn* :: *nat* ⇒ *nat*
obtain *nna* :: *nat* **where**
ff1: (*nna*, Γ, (*bdy*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧ (∀ *n*. ¬ *n* < *nna*

$\lor$ (*n*, Γ, (*bdy*, *Normal s*) # *xs*) $\notin$ *cptn-mod-nest-call*)
by (*meson CptnModNestCall.hyps*(*2*))
moreover
{ assume (*nn* (*nn* (*Suc nna*)), Γ, (*bdy*, *Normal s*) # *xs*) $\in$ *cptn-mod-nest-call*
then have $\neg$ *Suc* (*nn* (*nn* (*Suc nna*))) < *Suc nna*
using *ff1* **by** *blast*
then have (*nn* (*Suc nna*), Γ, (*LanguageCon.com.Call f*, *Normal s*) # (*bdy*, *Normal s*) # *xs*) $\in$ *cptn-mod-nest-call* $\longrightarrow$ ($\exists$ *n*. (*n*, Γ, (*LanguageCon.com.Call f*, *Normal s*) # (*bdy*, *Normal s*) # *xs*) $\in$ *cptn-mod-nest-call* $\land$
($\neg$ *nn n* < *n* $\lor$ (*nn n*, Γ, (*LanguageCon.com.Call f*, *Normal s*) # (*bdy*, *Normal s*) # *xs*) $\notin$ *cptn-mod-nest-call*))
using *ff1* **by** (*meson CptnModNestCall.hyps*(*3*) *CptnModNestCall.hyps*(*4*) *cptn-mod-nest-call.CptnModNestCall less-trans-Suc*) **}**
ultimately have $\exists$ *n*. (*n*, Γ, (*LanguageCon.com.Call f*, *Normal s*) # (*bdy*, *Normal s*) # *xs*) $\in$ *cptn-mod-nest-call* $\land$ ($\neg$ *nn n* < *n* $\lor$ (*nn n*, Γ, (*LanguageCon.com.Call f*, *Normal s*) # (*bdy*, *Normal s*) # *xs*) $\notin$ *cptn-mod-nest-call*)
by (*metis* (*no-types*) *CptnModNestCall.hyps*(*3*) *CptnModNestCall.hyps*(*4*) *cptn-mod-nest-call.CptnModNestCall elim-cptn-mod-nest-call-n*) **}**
then show *?thesis*
by *meson*
qed
next
case (*CptnModNestDynCom n* Γ *c s xs*) **thus** *?case*
by (*meson cptn-mod-nest-call.CptnModNestDynCom elim-cptn-mod-nest-call-n*)
next
case (*CptnModNestGuard n* Γ *c s xs g f*) **thus** *?case*
by (*meson cptn-mod-nest-call.CptnModNestGuard elim-cptn-mod-nest-call-n*)
next
case (*CptnModNestCatch1 n* Γ *P s xs zs Q*) **thus** *?case*
by (*metis* (*no-types*, *lifting*) *Catch-P-Not-finish cptn-mod-nest-call.CptnModNestCatch1 div-catch-nest*)
next
case (*CptnModNestCatch2 n* Γ *P s xs ys zs Q*)
then obtain *min-p* **where**
min-p:(*min-p*, Γ, (*P*, *s*) # *xs*) $\in$ *cptn-mod-nest-call* $\land$
($\forall$ *m*<*min-p*. (*m*, Γ, (*P*, *s*) # *xs*) $\notin$ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestCatch2*(*5*) **obtain** *min-q* **where**
min-q:(*min-q*, Γ, (*Skip*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*) $\in$ *cptn-mod-nest-call* $\land$
($\forall$ *m*<*min-q*. (*m*, Γ, (*Skip*, *snd* (*last* ((*P*, *s*) # *xs*))) # *ys*) $\notin$ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof(*cases min-p*$\geq$*min-q*)
case *True*
then have (*min-p*, Γ, (*Skip*, *snd* (*last* ((*P*,*s*) # *xs*))) # *ys*) $\in$ *cptn-mod-nest-call*
using *min-q* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-p*, Γ, (*Catch P Q*, *s*) # *zs*) $\in$ *cptn-mod-nest-call*
using *conjunct1*[*OF min-p*] *cptn-mod-nest-call.CptnModNestCatch2*[*of min-p* Γ *P s xs*]

CptnModNestCatch2(6) CptnModNestCatch2(3)
by *blast*
also have ∀ *m<min-p*. (*m*, Γ,(*Catch P Q*,*s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**fix** *m*
assume *min-m*:*m<min-p*
then have (*m*, Γ,(*Catch P Q*, *s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**assume** *ass*:(*m*, Γ, (*Catch P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
then obtain *xs′ s′ s″* **where**
m-cptn:(*m*, Γ, (*P*, *s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
catch-cond-nest zs Q xs′ P s s″ s′ Γ *m*
using
div-catch-nest[*of m* Γ (*Catch P Q*, *s*) # *zs*]
by *fastforce*
then have *catch-cond-nest zs Q xs′ P s s″ s′* Γ *m* **by** *auto*
then have *xs*=*xs′*
using *Catch-P-Ends-Skip*[*OF CptnModNestCatch2(6) CptnModNest-Catch2(3)*]
by *fastforce*
then have (*m*, Γ, (*P*,*s*) # *xs*) ∈ *cptn-mod-nest-call*
using *m-cptn* **by** *auto*
then have *False* **using** *min-p min-m* **by** *fastforce*
} **thus** *?thesis* **by** *auto*
qed
}**thus** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
next
case *False*
then have (*min-q*, Γ, (*P*, *s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have (*min-q*, Γ, (*Catch P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-q*] *cptn-mod-nest-call.CptnModNestCatch2*[*of min-q* Γ *P s xs*]
CptnModNestCatch2(6) CptnModNestCatch2(3)
by *blast*
also have ∀ *m<min-q*. (*m*, Γ,(*Catch P Q*,*s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**fix** *m*
assume *min-m*:*m<min-q*
then have (*m*, Γ,(*Catch P Q*, *s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{**assume** *ass*:(*m*, Γ, (*Catch P Q*, *s*) # *zs*) ∈ *cptn-mod-nest-call*
then obtain *xs′ s′ s″* **where**
m-cptn:(*m*, Γ, (*P*, *s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
catch-cond-nest zs Q xs′ P s s″ s′ Γ *m*
using
div-catch-nest[*of m* Γ (*Catch P Q*, *s*) # *zs*]

by *fastforce*
then have *catch-cond-nest zs Q xs′ P s s′′ s′ Γ m* **by** *auto*
then have *?thesis*
using *Catch-P-Ends-Skip*[*OF CptnModNestCatch2*(*6*) *CptnModNest-Catch2*(*3*)]
min-m min-q
by *blast*
} **thus** *?thesis* **by** *auto*
qed
}thus *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*CptnModNestCatch3 n Γ P s xs s′ Q ys zs*) **then obtain** *min-p* **where**
min-p:(*min-p*, Γ, (*P*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-p*. (*m*, Γ, (*P*, *Normal s*) # *xs*) ∉ *cptn-mod-nest-call*)
by *auto*
from *CptnModNestCatch3*(*6*) *CptnModNestCatch3*(*4*) **obtain** *min-q* **where**
min-q:(*min-q*, Γ, (*Q*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(∀ *m*<*min-q*. (*m*, Γ, (*Q*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∉ *cptn-mod-nest-call*)
by *auto*
thus *?case*
proof(*cases min-p*≥*min-q*)
case *True*
then have (*min-p*, Γ, (*Q*, *snd* (*last* ((*P*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-q* **using** *cptn-mod-nest-mono* **by** *blast*
then have (*min-p*, Γ, (*Catch P Q*, *Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
using *conjunct1*[*OF min-p*] *cptn-mod-nest-call.CptnModNestCatch3*[*of min-p Γ P s xs s′ Q ys zs*]
CptnModNestCatch3(*4*) *CptnModNestCatch3*(*3*) *CptnModNestCatch3*(*7*)
by *fastforce*
also have ∀ *m*<*min-p*. (*m*, Γ,(*Catch P Q*,*Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{fix *m*
assume *min-m*:*m*<*min-p*
then have (*m*, Γ,(*Catch P Q*, *Normal s*) # *zs*) ∉ *cptn-mod-nest-call*
proof −
{assume *ass*:(*m*, Γ, (*Catch P Q*,*Normal s*) # *zs*) ∈ *cptn-mod-nest-call*
then obtain *xs′ ns′ ns′′* **where**
m-cptn:(*m*, Γ, (*P*, *Normal s*) # *xs′*) ∈ *cptn-mod-nest-call* ∧
catch-cond-nest zs Q xs′ P (*Normal s*) *ns′′ ns′* Γ *m*
using
div-catch-nest[*of m* Γ (*Catch P Q*, *Normal s*) # *zs*]
by *fastforce*
then have *catch-cond-nest zs Q xs′ P* (*Normal s*) *ns′′ ns′* Γ *m* **by** *auto*

then have *xs=xs′*
using *Catch-P-Ends-Normal*[*OF CptnModNestCatch3*(*7*) *CptnModNest-Catch3*(*3*) *CptnModNestCatch3*(*4*)]
by *fastforce*
then have $(m, \Gamma, (P, Normal\ s) \# xs) \in$ *cptn-mod-nest-call*
using *m-cptn* **by** *auto*
then have *False* **using** *min-p min-m* **by** *fastforce*
} **thus** *?thesis* **by** *auto*
qed
}thus *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
next
case *False*
then have $(min\text{-}q, \Gamma, (P,\ Normal\ s) \# xs) \in$ *cptn-mod-nest-call*
using *min-p cptn-mod-nest-mono* **by** *force*
then have $(min\text{-}q, \Gamma, (Catch\ P\ Q, Normal\ s) \# zs) \in$ *cptn-mod-nest-call*
using *conjunct1*[*OF min-q*] *cptn-mod-nest-call.CptnModNestCatch3*[*of min-q* Γ *P s xs s′*]
CptnModNestCatch3(*4*) *CptnModNestCatch3*(*3*) *CptnModNestCatch3*(*7*)
by *blast*
also have $\forall m<min\text{-}q.\ (m, \Gamma,(Catch\ P\ Q,Normal\ s) \# zs) \notin$ *cptn-mod-nest-call*
proof −
{fix *m*
assume *min-m*:*m<min-q*
then have $(m, \Gamma,(Catch\ P\ Q, Normal\ s) \# zs) \notin$ *cptn-mod-nest-call*
proof −
{assume *ass*:$(m, \Gamma, (Catch\ P\ Q, Normal\ s) \# zs) \in$ *cptn-mod-nest-call*
then obtain *xs′ ns′ ns′′* **where**
m-cptn:$(m, \Gamma, (P, Normal\ s) \# xs') \in$ *cptn-mod-nest-call* ∧
catch-cond-nest zs Q xs′ P (*Normal s*) *ns′′ ns′* Γ *m*
using
div-catch-nest[*of m* Γ (*Catch P Q*, *Normal s*) # *zs*]
by *fastforce*
then have *catch-cond-nest zs Q xs′ P* (*Normal s*) *ns′′ ns′* Γ *m* **by** *auto*
then have *?thesis*
using *Catch-P-Ends-Normal*[*OF CptnModNestCatch3*(*7*) *CptnModNest-Catch3*(*3*) *CptnModNestCatch3*(*4*)]
min-m min-q
by (*metis last-length*)
} **thus** *?thesis* **by** *auto*
qed
}thus *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
qed

lemma *elim-cptn-mod-min-nest-call*:

assumes *a0*:*min-call n* Γ *cfg* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1* **and**
a2:(∀ *f*. *redex P* ≠ *Call f*) ∨
SmallStepCon.redex P = *LanguageCon.com.Call fn* ∧ Γ *fn* = *None* ∨
(*redex P* = *Call fn* ∧ (∀ *sa*. *s*≠*Normal sa*)) ∨
(*redex P* = *Call fn* ∧ *P*=*Q*)
shows *min-call n* Γ ((*Q*,*t*)#*cfg1*)
proof −
have *a0*: (*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a0': (∀ *m*<*n*. (*m*, Γ, *cfg*) ∉ *cptn-mod-nest-call*)
using *a0* **unfolding** *min-call-def* **by** *auto*
then have (*n*,Γ,(*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *a0 a1 elim-cptn-mod-nest-call-n* **by** *blast*
also have (∀ *m*<*n*. (*m*, Γ, (*Q*,*t*)#*cfg1*) ∉ *cptn-mod-nest-call*)
proof−
{ assume ¬(∀ *m*<*n*. (*m*, Γ, (*Q*,*t*)#*cfg1*) ∉ *cptn-mod-nest-call*)
then obtain *m* **where**
asm0:*m*<*n* **and**
asm1:(*m*, Γ, (*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
by *auto*
then have (*m*, Γ, *cfg*) ∈ *cptn-mod-nest-call*
using *a0 a1 a2 cptn-mod-nest-cptn-mod cptn-if-cptn-mod cptn-mod-nest-call.CptnModNestEnv*
cptn-elim-cases(*2*) *not-func-redex-cptn-mod-nest-n'*
by (*metis* (*no-types*, *lifting*) *mod-env-not-component*)

then have *False* **using** *a0' asm0* **by** *auto*
} thus *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed

lemma *elim-call-cptn-mod-min-nest-call*:
assumes *a0*:*min-call n* Γ *cfg* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1* **and**
a2:*P* = *Call f* ∧
Γ *f* = *Some Q* ∧ (∃ *sa*. *s*=*Normal sa*) ∧ *P*≠*Q*
shows *min-call* (*n*−*1*) Γ ((*Q*,*t*)#*cfg1*)
proof −
obtain *s'* **where** *a0*: (*n*,Γ,*cfg*) ∈ *cptn-mod-nest-call* **and**
a0': (∀ *m*<*n*. (*m*, Γ, *cfg*) ∉ *cptn-mod-nest-call*) **and**
a2': *s*= *Normal s'*
using *a0 a2* **unfolding** *min-call-def* **by** *auto*
then have (*n*−*1*,Γ,(*Q*,*t*)#*cfg1*) ∈ *cptn-mod-nest-call*
using *a1 a2 a2' elim-cptn-mod-nest-call-n-dec*[*of n* Γ *f s' cfg1 Q*]
by (*metis* (*no-types*, *lifting*) *SmallStepCon.redex.simps*(*7*) *call-f-step-not-s-eq-t-false*

cptn-elim-cases(*2*) *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod option.sel*)

thus *?thesis*
proof −
obtain *nn* :: (('*b*, '*a*, '*c*, '*d*) *LanguageCon.com* × ('*b*, '*c*) *xstate*) *list* ⇒
('*a* ⇒ ('*b*, '*a*, '*c*, '*d*) *LanguageCon.com option*) ⇒ *nat* ⇒ *nat* **where**
∀ *x0 x1 x2*. (∃ *v3*<*x2*. (*v3*, *x1*, *x0*) ∈ *cptn-mod-nest-call*) =
(*nn x0 x1 x2* < *x2* ∧ (*nn x0 x1 x2*, *x1*, *x0*) ∈ *cptn-mod-nest-call*)
by *moura*
then have *f1*: ∀ *n f ps*. (¬ *min-call n f ps* ∨ (*n*, *f*, *ps*) ∈ *cptn-mod-nest-call* ∧
(∀ *na*. ¬ *na* < *n* ∨ (*na*, *f*, *ps*) ∉ *cptn-mod-nest-call*)) ∧
(*min-call n f ps* ∨ (*n*, *f*, *ps*) ∉ *cptn-mod-nest-call* ∨
nn ps f n < *n* ∧ (*nn ps f n*, *f*, *ps*) ∈ *cptn-mod-nest-call*)
by (*meson min-call-def*)
then have *f2*: (*n*, Γ, (*P*, *s*) # (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call* ∧
(∀ *na*. ¬ *na* < *n* ∨ (*na*, Γ, (*P*, *s*) # (*Q*, *t*) # *cfg1*) ∉ *cptn-mod-nest-call*)
using *a1 assms*(*1*) **by** *blast*
obtain *bb* :: '*b* **where**
f3: *s* = *Normal bb*
using *a2* **by** *blast*
then have *f4*: (*LanguageCon.com.Call f*, *Normal bb*) = (*P*, *s*)
using *a2* **by** *blast*
have *f5*: *n* − *1* < *n*
using *f2* **by** (*metis* (*no-types*) *Suc-diff-Suc a2 diff-Suc-eq-diff-pred elim-cptn-mod-nest-call-n-greater-zero lessI minus-nat.diff-0*)
have *f6*: (*LanguageCon.com.Call f*, *Normal bb*) = (*P*, *s*)
using *f3 a2* **by** *blast*
have *f7*: *Normal bb* = *t*
using *f4 f2* **by** (*metis* (*no-types*) *SmallStepCon.redex.simps*(*7*) *a2*
call-f-step-not-s-eq-t-false cptn-elim-cases(*2*)
cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod)
have (*nn* ((*Q*, *t*) # *cfg1*) Γ (*n* − *1*), Γ, (*Q*, *Normal bb*) # *cfg1*) ∈ *cptn-mod-nest-call*
⟶
(*Suc* (*nn* ((*Q*, *t*) # *cfg1*) Γ (*n* − *1*)), Γ,
(*LanguageCon.com.Call f*, *Normal bb*) # (*Q*, *Normal bb*) # *cfg1*) ∈
cptn-mod-nest-call
using *a2 cptn-mod-nest-call.CptnModNestCall* **by** *fastforce*
then show *?thesis*
using *f7 f6 f5 f2 f1* ‹(*n* − *1*, Γ, (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*›
less-trans-Suc **by** *blast*
qed

qed

lemma *redex-not-call-seq-catch*:
assumes *a0*:*redex P* = *Call f* ∧ *P*≠*Call f*
shows ∃ *p1 p2*. *P* = *Seq p1 p2* ∨ *P* = *Catch p1 p2*
using *a0* **unfolding** *min-call-def*
proof(*induct P*)
qed(*fastforce*+)

lemma *skip-all-skip*:
  assumes *a0*:(Γ,*cfg*)∈*cptn* **and**
          a1:*cfg* = (*Skip*,*s*)#*cfg1*
  shows ∀ *i*<*length cfg*. *fst*(*cfg*!*i*) = *Skip*
using *a0 a1*
proof(*induct cfg1 arbitrary*:*cfg s*)
  case *Nil* **thus** *?case* **by** *auto*
next
  case (*Cons x xs*)
  then obtain *s'* **where** *x*:*x* = (*Skip*,*s'*)
    by (*metis CptnMod-elim-cases*(*1*) *cptn-eq-cptn-mod-set stepc-elim-cases*(*1*))
  moreover have *cptn*:(Γ,*x*#*xs*)∈*cptn*
    using *Cons.prems*(*1*) *Cons.prems*(*2*) *cptn-dest-pair* **by** *blast*
  moreover have
    xs:*x* # *xs* = (*LanguageCon.com.Skip*, *s'*) # *xs* **using** *x* **by** *auto*
  ultimately show *?case* **using** *Cons*(*1*)[*OF cptn xs*] *Cons*(*3*)
    using *diff-Suc-1 fstI length-Cons less-Suc-eq-0-disj nth-Cons'* **by** *auto*
qed

lemma *skip-all-skip-throw*:
  assumes *a0*:(Γ,*cfg*)∈*cptn* **and**
          a1:*cfg* = (*Throw*,*s*)#*cfg1*
  shows ∀ *i*<*length cfg*. *fst*(*cfg*!*i*) = *Skip* ∨ *fst*(*cfg*!*i*) = *Throw*
using *a0 a1*
proof(*induct cfg1 arbitrary*:*cfg s*)
  case *Nil* **thus** *?case* **by** *auto*
next
  case (*Cons x xs*)
  then obtain *s'* **where** *x*:*x* = (*Skip*,*s'*) ∨ *x* = (*Throw*, *s'*)
    by (*metis CptnMod-elim-cases*(*10*) *cptn-eq-cptn-mod-set*)
  then have *cptn*:(Γ,*x*#*xs*)∈*cptn*
    using *Cons.prems*(*1*) *Cons.prems*(*2*) *cptn-dest-pair* **by** *blast*
  show *?case* **using** *x*
  proof
    assume *x*=(*Skip*,*s'*) **thus** *?thesis* **using** *skip-all-skip Cons*(*3*)
    using *cptn fstI length-Cons less-Suc-eq-0-disj nth-Cons' nth-Cons-Suc skip-all-skip*

      by *fastforce*
  next
    assume *x*:*x*=(*Throw*,*s'*)
    moreover have *cptn*:(Γ,*x*#*xs*)∈*cptn*
      using *Cons.prems*(*1*) *Cons.prems*(*2*) *cptn-dest-pair* **by** *blast*
    moreover have
      xs:*x* # *xs* = (*LanguageCon.com.Throw*, *s'*) # *xs* **using** *x* **by** *auto*
    ultimately show *?case* **using** *Cons*(*1*)[*OF cptn xs*] *Cons*(*3*)
    using *diff-Suc-1 fstI length-Cons less-Suc-eq-0-disj nth-Cons'* **by** *auto*
  qed
qed

lemma *skip-min-nested-call-0*:
  assumes *a0*:*min-call n* Γ *cfg* **and**
          a1:*cfg* = (*Skip*,*s*)#*cfg1*
  shows *n=0*
proof −
  have *asm0*:(*n*, Γ, *cfg*) ∈ *cptn-mod-nest-call* **and**
       asm1:(∀ *m*<*n*. (*m*, Γ, *cfg*) ∉ *cptn-mod-nest-call*)
       using *a0* **unfolding** *min-call-def* **by** *auto*
  show *?thesis* **using** *a1 asm0 asm1*
  proof (*induct cfg1 arbitrary*: *cfg s n*)
    case *Nil* **thus** *?case*
      using *cptn-mod-nest-call.CptnModNestOne neq0-conv* **by** *blast*
  next
    case (*Cons x xs*)
      then obtain *Q s'* **where** *cfg*:*cfg* = (*LanguageCon.com.Skip*, *s*) # (*Q*,*s'*) # *xs* **by** *force*
     then have *min-call*:*min-call n* Γ *cfg* **using** *Cons* **unfolding** *min-call-def* **by** *auto*
      then have (∀ *f*. *SmallStepCon.redex Skip* ≠ *LanguageCon.com.Call f*) **by** *auto*
      then have *min-call n* Γ ((*Q*, *s'*)#*xs*)
        using *elim-cptn-mod-min-nest-call*[*OF min-call cfg*] *cfg*
        by *simp*
      thus *?case* **using** *Cons cfg* **unfolding** *min-call-def*
      proof −
        assume *a1*: (*n*, Γ, (*Q*, *s'*) # *xs*) ∈ *cptn-mod-nest-call* ∧ (∀ *m*<*n*. (*m*, Γ, (*Q*, *s'*) # *xs*) ∉ *cptn-mod-nest-call*)
        have *LanguageCon.com.Skip* = *Q*
          by (*metis* (*no-types*) ‹(*n*, Γ, *cfg*) ∈ *cptn-mod-nest-call*› *cfg cptn-dest1-pair cptn-if-cptn-mod cptn-mod-nest-cptn-mod fst-conv last.simps last-length length-Cons lessI not-Cons-self2 skip-all-skip*)
        then show *?thesis*
          using *a1* **by** (*meson Cons.hyps*)
      qed
  qed
qed

lemma *throw-min-nested-call-0*:
  assumes *a0*:*min-call n* Γ *cfg* **and**
          a1:*cfg* = (*Throw*,*s*)#*cfg1*
  shows *n=0*
proof −
  have *asm0*:(*n*, Γ, *cfg*) ∈ *cptn-mod-nest-call* **and**
       asm1:(∀ *m*<*n*. (*m*, Γ, *cfg*) ∉ *cptn-mod-nest-call*)
       using *a0* **unfolding** *min-call-def* **by** *auto*
  show *?thesis* **using** *a1 asm0 asm1*
  proof (*induct cfg1 arbitrary*: *cfg s n*)
    case *Nil* **thus** *?case*

using *cptn-mod-nest-call.CptnModNestOne neq0-conv* **by** *blast*
next
case (*Cons x xs*)
then obtain s' **where** x:x = (*Skip*,s') ∨ x = (*Throw*, s')
using *CptnMod-elim-cases*(*10*) *cptn-eq-cptn-mod-set*
by (*metis cptn-mod-nest-cptn-mod*)
then obtain Q **where** *cfg*:*cfg* = (*LanguageCon.com.Throw*, s) # (Q,s') # *xs*
using *Cons* **by** *force*
then have *min-call*:*min-call n* Γ *cfg* **using** *Cons* **unfolding** *min-call-def* **by** *auto*
then have (∀ *f*. *SmallStepCon.redex Skip* ≠ *LanguageCon.com.Call f*) **by** *auto*
then have *min-call'*:*min-call n* Γ ((*Q*, s')#*xs*)
using *elim-cptn-mod-min-nest-call*[*OF min-call cfg*] *cfg*
by *simp*
from x **show** *?case*
proof
assume x=(*Skip*,s')
thus *?thesis* **using** *skip-min-nested-call-0 min-call' Cons*(*2*) *cfg* **by** *fastforce*
next
assume x=(*Throw*,s')
thus *?thesis* **using** *Cons*(*1*,*2*) *min-call' cfg* **unfolding** *min-call-def*
by *blast*
qed
qed
qed

function to calculate that there is not any subsequent where the nested call is n

definition *cond-seq-1*
where
cond-seq-1 n Γ *c1 s xs c2 zs ys* ≡ ((*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
fst(*last*((*c1*,*s*)#*xs*)) = *Skip* ∧
(*n*,Γ,((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift c2*) *xs*)@((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*))

definition *cond-seq-2*
where
cond-seq-2 n Γ *c1 s xs c2 zs ys* s' s'' ≡ *s*= *Normal* s'' ∧
(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
fst(*last* ((*c1*, *s*)#*xs*)) = *Throw* ∧
snd(*last* ((*c1*, *s*)#*xs*)) = *Normal* s' ∧
(*n*,Γ,(*Throw*,*Normal* s')#*ys*) ∈ *cptn-mod-nest-call* ∧
zs=(*map* (*lift c2*) *xs*)@((*Throw*,*Normal* s')#*ys*)

definition *cond-catch-1*
where
cond-catch-1 n Γ *c1 s xs c2 zs ys* ≡ ((*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧

$fst(last((c1,s)\#xs)) = Skip \wedge$
$(n,\Gamma,((Skip,\ snd(last\ ((c1,\ s)\#xs)))\#ys)) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\wedge$
$zs=(map\ (lift\text{-}catch\ c2)\ xs)@((Skip,\ snd(last\ ((c1,\ s)\#xs)))\#ys))$

definition *cond-catch-2*
where
$cond\text{-}catch\text{-}2\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys\ s'\ s'' \equiv s= Normal\ s'' \wedge$
$(n,\Gamma,\ (c1,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$fst(last\ ((c1,\ s)\#xs)) = Throw \wedge$
$snd(last\ ((c1,\ s)\#xs)) = Normal\ s' \wedge$
$(n,\Gamma,(c2,Normal\ s')\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$zs=(map\ (lift\text{-}catch\ c2)\ xs)@((c2,Normal\ s')\#ys)$

fun $biggest\text{-}nest\text{-}call :: ('s,'p,'f,'e)com \Rightarrow$
$('s,'f)\ xstate \Rightarrow$
$(('s,'p,'f,'e)\ config)\ list \Rightarrow$
$('s,'p,'f,'e)\ body \Rightarrow$
$nat \Rightarrow bool$
where
$biggest\text{-}nest\text{-}call\ (Seq\ c1\ c2)\ s\ zs\ \Gamma\ n\ =$
$(if\ (\exists xs.\ ((min\text{-}call\ n\ \Gamma\ ((c1,s)\#xs)) \wedge (zs=map\ (lift\ c2)\ xs)))\ then$
$let\ xsa = (SOME\ xs.\ (min\text{-}call\ n\ \Gamma\ ((c1,s)\#xs)) \wedge (zs=map\ (lift\ c2)\ xs))\ in$
$(biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n)$
$else\ if\ (\exists xs\ ys.\ cond\text{-}seq\text{-}1\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys)\ then$
$let\ xsa = (SOME\ xs.\ \exists ys.\ cond\text{-}seq\text{-}1\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys);$
$ysa = (SOME\ ys.\ cond\text{-}seq\text{-}1\ n\ \Gamma\ c1\ s\ xsa\ c2\ zs\ ys)\ in$
$if\ (min\text{-}call\ n\ \Gamma\ ((c2,\ snd(last\ ((c1,\ s)\#xsa)))\#ysa))\ then\ True$
$else\ (biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n)$
$else\ let\ xsa = (SOME\ xs.\ \exists ys\ s'\ s''.\ cond\text{-}seq\text{-}2\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys\ s'\ s'')\ in$
$(biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n))$
$|biggest\text{-}nest\text{-}call\ (Catch\ c1\ c2)\ s\ zs\ \Gamma\ n\ =$
$(if\ (\exists xs.\ ((min\text{-}call\ n\ \Gamma\ ((c1,s)\#xs)) \wedge (zs=map\ (lift\text{-}catch\ c2)\ xs)))\ then$
$let\ xsa = (SOME\ xs.\ (min\text{-}call\ n\ \Gamma\ ((c1,s)\#xs)) \wedge (zs=map\ (lift\text{-}catch\ c2)\ xs))$
in
$(biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n)$
$else\ if\ (\exists xs\ ys.\ cond\text{-}catch\text{-}1\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys)\ then$
$let\ xsa = (SOME\ xs.\ \exists ys.\ cond\text{-}catch\text{-}1\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys)\ in$
$(biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n)$
$else\ let\ xsa = (SOME\ xs.\ \exists ys\ s'\ s''.\ cond\text{-}catch\text{-}2\ n\ \Gamma\ c1\ s\ xs\ c2\ zs\ ys\ s'\ s'');$
$ysa = (SOME\ ys.\ \exists s'\ s''.\ cond\text{-}catch\text{-}2\ n\ \Gamma\ c1\ s\ xsa\ c2\ zs\ ys\ s'\ s'')\ in$
$if\ (min\text{-}call\ n\ \Gamma\ ((c2,\ snd(last\ ((c1,\ s)\#xsa)))\#ysa))\ then\ True$
$else\ (biggest\text{-}nest\text{-}call\ c1\ s\ xsa\ \Gamma\ n))$
$|biggest\text{-}nest\text{-}call\ \text{-}\ \text{-}\ \text{-}\ \text{-}\ \text{-} = False$

lemma *min-call-less-eq-n*:
$(n,\Gamma,\ (c1,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call \Longrightarrow$
$(n,\Gamma,(c2,\ snd(last\ ((c1,\ s)\#xs)))\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call \Longrightarrow$

min-call p Γ ((*c1*, *s*)#*xs*) ∧ *min-call q* Γ ((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ⟹
p≤*n* ∧ *q*≤*n*
unfolding *min-call-def*
using *le-less-linear* **by** *blast*

lemma *min-call-seq-less-eq-n'*:
(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ⟹
min-call p Γ ((*c1*, *s*)#*xs*) ⟹
p≤*n*
unfolding *min-call-def*
using *le-less-linear* **by** *blast*

lemma *min-call-seq2*:
min-call n Γ ((*Seq c1 c2*,*s*)#*zs*) ⟹
(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ⟹
fst(*last* ((*c1*, *s*)#*xs*)) = *Skip* ⟹
(*n*,Γ,(*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* ⟹
zs=(*map* (*lift c2*) *xs*)@((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ⟹
min-call n Γ ((*c1*, *s*)#*xs*) ∨ *min-call n* Γ ((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)

proof −
assume *a0*:*min-call n* Γ ((*Seq c1 c2*,*s*)#*zs*) **and**
a1:(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
a2:*fst*(*last* ((*c1*, *s*)#*xs*)) = *Skip* **and**
a3:(*n*,Γ,(*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* **and**
a4:*zs*=(*map* (*lift c2*) *xs*)@((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)
then obtain *p q* **where** *min-calls*:
min-call p Γ ((*c1*, *s*)#*xs*) ∧ *min-call q* Γ ((*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)
using *a1 a3 minimum-nest-call* **by** *blast*
then have *p-q*:*p*≤*n* ∧ *q*≤*n* **using** *a0 a1 a3 a4 min-call-less-eq-n* **by** *blast*
{
assume *ass0*:*p*<*n* ∧ *q* <*n*
then have (*p*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
(*q*,Γ,(*c2*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have *?thesis*
proof (*cases p*≤*q*)
case *True*
then have *q-cptn-c1*:(*q*, Γ, (*c1*, *s*) # *xs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-mono min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(*q*, Γ, (*c2*, *snd* (*last* ((*c1*, *s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have (*q*,Γ,((*Seq c1 c2*,*s*)#*zs*)) ∈*cptn-mod-nest-call*
using *True min-calls a2 a4 CptnModNestSeq2*[*OF q-cptn-c1 a2 q-cptn-c2 a4*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
next

case *False*
then have *q-cptn-c1*:(p, Γ, ($c1$, s) # xs) $\in$ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(p, Γ, ($c2$, *snd* (*last* (($c1$, s) # xs))) # ys) $\in$ *cptn-mod-nest-call*
using *min-calls False* **unfolding** *min-call-def*
by (*metis* (*no-types*, *lifting*) *cptn-mod-nest-mono2 not-less*)
then have (p,Γ,((*Seq c1 c2*,s)#zs)) $\in$*cptn-mod-nest-call*
using *False min-calls a2 a4 CptnModNestSeq2*[*OF q-cptn-c1 a2 q-cptn-c2 a4*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
qed
}note *l*=*this*
{
assume *ass0*:$p \geq n \vee q \geq n$
then have *?thesis* **using** *p-q min-calls* **by** *fastforce*
}
thus *?thesis* **using** *l* **by** *fastforce*
qed

lemma *min-call-seq3*:
min-call n Γ ((*Seq c1 c2*,s)#zs) $\Longrightarrow$
s= *Normal* s'' $\Longrightarrow$
(n,Γ, ($c1$, s)#xs) $\in$ *cptn-mod-nest-call* $\Longrightarrow$
fst(*last* (($c1$, s)#xs)) = *Throw* $\Longrightarrow$
snd(*last* (($c1$, s)#xs)) = *Normal* s' $\Longrightarrow$
(n,Γ,(*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys) $\in$ *cptn-mod-nest-call* $\Longrightarrow$
zs=(*map* (*lift c2*) xs)@((*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys) $\Longrightarrow$
min-call n Γ (($c1$, s)#xs)

proof –
assume *a0*:*min-call n* Γ ((*Seq c1 c2*,s)#zs) **and**
a0':s= *Normal* s'' **and**
a1:(n,Γ, ($c1$, s)#xs) $\in$ *cptn-mod-nest-call* **and**
a2:*fst*(*last* (($c1$, s)#xs)) = *Throw* **and**
a2':*snd*(*last* (($c1$, s)#xs)) = *Normal* s' **and**
a3:(n,Γ,(*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys) $\in$ *cptn-mod-nest-call* **and**
a4:zs=(*map* (*lift c2*) xs)@((*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys)
then obtain p **where** *min-calls*:
min-call p Γ (($c1$, s)#xs) $\wedge$ *min-call 0* Γ ((*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys)
using *a1 a3 minimum-nest-call throw-min-nested-call-0* **by** *metis*
then have *p-q*:$p \leq n \wedge 0 \leq n$ **using** *a0 a1 a3 a4 min-call-less-eq-n* **by** *blast*
{
assume *ass0*:$p<n \wedge 0 <n$
then have (p,Γ, ($c1$, s)#xs) $\in$ *cptn-mod-nest-call* **and**
(*0*,Γ,(*Throw*, *snd*(*last* (($c1$, s)#xs)))#ys) $\in$ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have *?thesis*

proof (*cases* $p \le 0$)
case *True*
then have *q-cptn-c1*:(0, Γ, (*c1*, *Normal* s'') # *xs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-mono min-calls a0'* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(0, Γ, (*Throw*, *snd* (*last* ((*c1*, *s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have (0,Γ,((*Seq c1 c2*,*s*)#*zs*)) ∈*cptn-mod-nest-call*
using *True min-calls a2 a4 a2' a0' CptnModNestSeq3*[*OF q-cptn-c1*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
next
case *False*
then have *q-cptn-c1*:(*p*, Γ, (*c1*, *Normal* s'') # *xs*) ∈ *cptn-mod-nest-call*
using *min-calls a0'* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(*p*, Γ, (*Throw*, *snd* (*last* ((*c1*, *s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-calls False* **unfolding** *min-call-def*
by (*metis* (*no-types*, *lifting*) *cptn-mod-nest-mono2 not-less*)
then have (*p*,Γ,((*Seq c1 c2*,*s*)#*zs*)) ∈*cptn-mod-nest-call*
using *False min-calls a2 a4 a0' a2' CptnModNestSeq3*[*OF q-cptn-c1*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
qed
}note *l=this*
{
assume *ass0*:$p \ge n \lor 0 \ge n$
then have *?thesis* **using** *p-q min-calls* **by** *fastforce*
}
thus *?thesis* **using** *l* **by** *fastforce*
qed

lemma *min-call-catch2*:
min-call n Γ ((*Catch c1 c2*,*s*)#*zs*) ⟹
(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ⟹
fst(*last* ((*c1*, *s*)#*xs*)) = *Skip* ⟹
(*n*,Γ,(*Skip*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* ⟹
zs=(*map* (*lift-catch c2*) *xs*)@((*Skip*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ⟹
min-call n Γ ((*c1*, *s*)#*xs*)

proof −
assume *a0*:*min-call n* Γ ((*Catch c1 c2*,*s*)#*zs*) **and**
a1:(*n*,Γ, (*c1*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
a2:*fst*(*last* ((*c1*, *s*)#*xs*)) = *Skip* **and**
a3:(*n*,Γ,(*Skip*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* **and**
a4:*zs*=(*map* (*lift-catch c2*) *xs*)@((*Skip*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)
then obtain *p* **where** *min-calls*:
min-call p Γ ((*c1*, *s*)#*xs*) ∧ *min-call 0* Γ ((*Skip*, *snd*(*last* ((*c1*, *s*)#*xs*)))#*ys*)
using *a1 a3 minimum-nest-call skip-min-nested-call-0* **by** *metis*

then have *p-q*:$p \le n \wedge 0 \le n$ **using** *a0 a1 a3 a4 min-call-less-eq-n* **by** *blast*
{
assume *ass0*:$p < n \wedge 0 < n$
then have $(p,\Gamma, (c1, s)\#xs) \in$ *cptn-mod-nest-call* **and**
$(0,\Gamma,(Skip, snd(last ((c1, s)\#xs)))\#ys) \in$ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have *?thesis*
proof (*cases* $p \le 0$)
case *True*
then have *q-cptn-c1*:$(0, \Gamma, (c1, s) \# xs) \in$ *cptn-mod-nest-call*
using *cptn-mod-nest-mono min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:$(0, \Gamma, (Skip, snd (last ((c1, s) \# xs))) \# ys) \in$ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have $(0,\Gamma,((Catch\ c1\ c2,s)\#zs)) \in$*cptn-mod-nest-call*
using *True min-calls a2 a4 CptnModNestCatch2*[*OF q-cptn-c1*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
next
case *False*
then have *q-cptn-c1*:$(p, \Gamma, (c1, s) \# xs) \in$ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:$(p, \Gamma, (Skip, snd (last ((c1, s) \# xs))) \# ys) \in$ *cptn-mod-nest-call*
using *min-calls False* **unfolding** *min-call-def*
by (*metis* (*no-types*, *lifting*) *cptn-mod-nest-mono2 not-less*)
then have $(p,\Gamma,((Catch\ c1\ c2,s)\#zs)) \in$*cptn-mod-nest-call*
using *False min-calls a2 a4 CptnModNestCatch2*[*OF q-cptn-c1*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
qed
}note *l=this*
{
assume *ass0*:$p \ge n \vee 0 \ge n$
then have *?thesis* **using** *p-q min-calls* **by** *fastforce*
}
thus *?thesis* **using** *l* **by** *fastforce*
qed

lemma *min-call-catch-less-eq-n*:
$(n,\Gamma, (c1, Normal\ s)\#xs) \in$ *cptn-mod-nest-call* $\Longrightarrow$
$(n,\Gamma,(c2, snd(last ((c1, Normal\ s)\#xs)))\#ys) \in$ *cptn-mod-nest-call* $\Longrightarrow$
min-call $p\ \Gamma\ ((c1, Normal\ s)\#xs) \wedge$ *min-call* $q\ \Gamma\ ((c2, snd(last ((c1, Normal\ s)\#xs)))\#ys) \Longrightarrow$
$p \le n \wedge q \le n$
unfolding *min-call-def*
using *le-less-linear* **by** *blast*

lemma *min-call-catch3*:

min-call n Γ ((*Catch c1 c2*,*Normal s*)#*zs*) ⟹
(*n*,Γ, (*c1*, *Normal s*)#*xs*) ∈ *cptn-mod-nest-call* ⟹
fst(*last* ((*c1*, *Normal s*)#*xs*)) = *Throw* ⟹
snd(*last* ((*c1*, *Normal s*)#*xs*)) = *Normal s′* ⟹
(*n*,Γ,(*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* ⟹
zs=(*map* (*lift-catch c2*) *xs*)@((*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*) ⟹
min-call n Γ ((*c1*, *Normal s*)#*xs*) ∨ *min-call n* Γ ((*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*)

proof −
assume *a0*:*min-call n* Γ ((*Catch c1 c2*,*Normal s*)#*zs*) **and**
a1:(*n*,Γ, (*c1*, *Normal s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
a2:*fst*(*last* ((*c1*, *Normal s*)#*xs*)) = *Throw* **and**
a2′:*snd*(*last* ((*c1*, *Normal s*)#*xs*)) = *Normal s′* **and**
a3:(*n*,Γ,(*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call* **and**
a4:*zs*=(*map* (*lift-catch c2*) *xs*)@((*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*)
then obtain *p q* **where** *min-calls*:
min-call p Γ ((*c1*, *Normal s*)#*xs*) ∧ *min-call q* Γ ((*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*)
using *a1 a3 minimum-nest-call* **by** *blast*
then have *p-q*:*p*≤*n* ∧ *q*≤*n*
using *a1 a2 a2′ a3 a4 min-call-less-eq-n* **by** *blast*
{
assume *ass0*:*p*<*n* ∧ *q* <*n*
then have (*p*,Γ, (*c1*, *Normal s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
(*q*,Γ,(*c2*, *snd*(*last* ((*c1*, *Normal s*)#*xs*)))#*ys*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have *?thesis*
proof (*cases p*≤*q*)
case *True*
then have *q-cptn-c1*:(*q*, Γ, (*c1*,*Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *cptn-mod-nest-mono min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(*q*, Γ, (*c2*, *snd* (*last* ((*c1*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def* **by** *auto*
then have (*q*,Γ,((*Catch c1 c2*, *Normal s*)#*zs*)) ∈*cptn-mod-nest-call*
using *True min-calls a2 a2′ a4 CptnModNestCatch3*[*OF q-cptn-c1 a2 a2′ q-cptn-c2 a4*]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
next
case *False*
then have *q-cptn-c1*:(*p*, Γ, (*c1*, *Normal s*) # *xs*) ∈ *cptn-mod-nest-call*
using *min-calls* **unfolding** *min-call-def*
by *blast*
have *q-cptn-c2*:(*p*, Γ, (*c2*, *snd* (*last* ((*c1*, *Normal s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call*
using *min-calls False* **unfolding** *min-call-def*

by (*metis* (*no-types*, *lifting*) *cptn-mod-nest-mono2 not-less*)
then have (p,Γ,((*Catch c1 c2*,*Normal s*)#*zs*)) $\in$*cptn-mod-nest-call*
using *False min-calls a2 a4 CptnModNestCatch3*[*OF q-cptn-c1 a2 a2′*
q-cptn-c2 a4]
by *auto*
thus *?thesis* **using** *ass0 a0* **unfolding** *min-call-def* **by** *auto*
qed
}note *l*=*this*
{
assume *ass0*:$p \geq n \vee q \geq n$
then have *?thesis* **using** *p-q min-calls* **by** *fastforce*
}
thus *?thesis* **using** *l* **by** *fastforce*
qed

lemma *min-call-seq-c1-not-finish*:
min-call n Γ *cfg* $\Longrightarrow$
cfg = (*LanguageCon.com.Seq P0 P1*, *s*) # (*Q*, *t*) # *cfg1* $\Longrightarrow$
(*n*, Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* $\Longrightarrow$
(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs* $\Longrightarrow$
min-call n Γ ((*P0*, *s*)#*xs*)

proof −
assume *a0*:*min-call n* Γ *cfg* **and**
a1: *cfg* = (*LanguageCon.com.Seq P0 P1*, *s*) # (*Q*, *t*) # *cfg1* **and**
a2:(*n*, Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* **and**
a3:(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*
then have (*n*, Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* **using** *a2* **by** *auto*
moreover have $\forall m<n.$ (*m*, Γ,(*P0*, *s*)#*xs*) $\notin$ *cptn-mod-nest-call*
proof−
{fix *m*
assume *ass*:$m<n$
{ assume *ass1*:(*m*, Γ, (*P0*, *s*) # *xs*) $\in$ *cptn-mod-nest-call*
then have (*m*,Γ,*cfg*) $\in$ *cptn-mod-nest-call*
using *a1 a3 CptnModNestSeq1*[*OF ass1*] **by** *auto*
then have *False* **using** *ass a0* **unfolding** *min-call-def* **by** *auto*
}
then have (*m*, Γ, (*P0*, *s*) # *xs*) $\notin$ *cptn-mod-nest-call* **by** *auto*
} then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed

lemma *min-call-seq-not-finish*:
min-call n Γ ((*P0*, *s*)#*xs*) $\Longrightarrow$
cfg = (*LanguageCon.com.Seq P0 P1*, *s*) # *cfg1* $\Longrightarrow$
cfg1 = *map* (*lift P1*) *xs* $\Longrightarrow$
min-call n Γ *cfg*

proof –
  assume *a0*:*min-call n* Γ ((*P0*, *s*)#*xs*) **and**
      a1: *cfg* = (*LanguageCon.com.Seq P0 P1*, *s*) # *cfg1* **and**
      a2: *cfg1* = *map* (*lift P1*) *xs*
  then have (*n*, Γ,*cfg*) ∈ *cptn-mod-nest-call*
  using *a0 a1 a2 CptnModNestSeq1*[*of n* Γ *P0 s xs cfg1 P1*] **unfolding** *min-call-def*

  by *auto*
  moreover have ∀ *m*<*n*. (*m*, Γ,*cfg*) ∉ *cptn-mod-nest-call*
  proof–
    {**fix** *m*
    assume *ass*:*m*<*n*
    { **assume** *ass1*:(*m*, Γ, *cfg*) ∈ *cptn-mod-nest-call*
      then have (*m*,Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call*
        using *a1 a2* **by** (*metis* (*no-types*) *Seq-P-Not-finish div-seq-nest*)
      then have *False* **using** *ass a0* **unfolding** *min-call-def* **by** *auto*
    }
    then have (*m*, Γ, *cfg*) ∉ *cptn-mod-nest-call* **by** *auto*
    } **then show** *?thesis* **by** *auto*
  qed
  ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed

lemma *min-call-catch-c1-not-finish*:
  min-call n Γ *cfg* ⟹
  cfg = (*LanguageCon.com.Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1* ⟹
  (*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ⟹
  (*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* ⟹
  min-call n Γ ((*P0*, *s*)#*xs*)

proof –
  assume *a0*:*min-call n* Γ *cfg* **and**
      a1: *cfg* = (*LanguageCon.com.Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1* **and**
      a2:(*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
      a3:(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs*
  then have (*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **using** *a2* **by** *auto*
  moreover have ∀ *m*<*n*. (*m*, Γ,(*P0*, *s*)#*xs*) ∉ *cptn-mod-nest-call*
  proof–
    {**fix** *m*
    assume *ass*:*m*<*n*
    { **assume** *ass1*:(*m*, Γ, (*P0*, *s*) # *xs*) ∈ *cptn-mod-nest-call*
      then have (*m*,Γ,*cfg*) ∈ *cptn-mod-nest-call*
        using *a1 a3 CptnModNestCatch1*[*OF ass1*] **by** *auto*
      then have *False* **using** *ass a0* **unfolding** *min-call-def* **by** *auto*
    }
    then have (*m*, Γ, (*P0*, *s*) # *xs*) ∉ *cptn-mod-nest-call* **by** *auto*
    } **then show** *?thesis* **by** *auto*
  qed
  ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*

qed

lemma *min-call-catch-not-finish*:
  min-call n Γ $((P0, s)\#xs) \Longrightarrow$
  $cfg = (LanguageCon.com.Catch\ P0\ P1, s)\ \#\ \ cfg1 \Longrightarrow$
  $cfg1 = map\ (lift\text{-}catch\ P1)\ xs \Longrightarrow$
  min-call n Γ *cfg*

proof −
 assume *a0*:*min-call* n Γ $((P0, s)\#xs)$ **and**
   a1: $cfg = (Catch\ P0\ P1, s)\ \#\ \ cfg1$ **and**
   a2: $cfg1 = map\ (lift\text{-}catch\ P1)\ xs$
 then have $(n, \Gamma, cfg) \in cptn\text{-}mod\text{-}nest\text{-}call$
   using *a0 a1 a2 CptnModNestCatch1*[*of* n Γ *P0 s xs cfg1 P1*] **unfolding** *min-call-def*
  by *auto*
 moreover have $\forall m<n.\ (m, \Gamma, cfg) \notin cptn\text{-}mod\text{-}nest\text{-}call$
 proof−
  {**fix** m
  assume *ass*:$m<n$
  { **assume** *ass1*:$(m, \Gamma, cfg) \in cptn\text{-}mod\text{-}nest\text{-}call$
   then have $(m, \Gamma, (P0, s)\#xs) \in\ cptn\text{-}mod\text{-}nest\text{-}call$
    using *a1 a2* **by** (*metis* (*no-types*) *Catch-P-Not-finish div-catch-nest*)
   then have *False* **using** *ass a0* **unfolding** *min-call-def* **by** *auto*
  }
  then have $(m, \Gamma, cfg) \notin cptn\text{-}mod\text{-}nest\text{-}call$ **by** *auto*
  } **then show** *?thesis* **by** *auto*
 qed
 ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed

lemma *seq-xs-no-empty*: **assumes**
  seq:*seq-cond-nest* $((Q,t)\#cfg1)$ *P1 xs P0 s* s'' s' Γ n **and**
  cfg:$cfg = (LanguageCon.com.Seq\ P0\ P1, s)\ \#\ (Q, t)\ \#\ cfg1$ **and**
  a0:$SmallStepCon.redex\ (LanguageCon.com.Seq\ P0\ P1) = LanguageCon.com.Call\ f$
  shows$\exists\ Q'\ xs'.\ Q=Seq\ Q'\ P1 \land xs=(Q',t)\#xs'$
using *seq*
unfolding *lift-def seq-cond-nest-def*
proof
  assume $(Q, t)\ \#\ cfg1 = map\ (\lambda(P, s).\ (LanguageCon.com.Seq\ P\ P1, s))\ xs$
  thus *?thesis* **by** *auto*
next
 assume $fst\ (((P0, s)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Skip\ \land$
   $(\exists\ ys.\ (n, \Gamma, (P1, snd\ (((P0, s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\land$
    $(Q, t)\ \#\ cfg1 =$
    $map\ (\lambda(P, s).\ (LanguageCon.com.Seq\ P\ P1, s))\ xs\ @$
    $(P1, snd\ (((P0, s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \lor$

```
      fst (((P0, s) # xs) ! length xs) = LanguageCon.com.Throw ∧
      snd (last ((P0, s) # xs)) = Normal s' ∧
      s = Normal s'' ∧
    (∃ ys. (n, Γ, (LanguageCon.com.Throw, Normal s') # ys) ∈ cptn-mod-nest-call
∧
          (Q, t) # cfg1 =
          map (λ(P, s). (LanguageCon.com.Seq P P1, s)) xs @
          (LanguageCon.com.Throw, Normal s') # ys)
  thus ?thesis
  proof
    assume ass:fst (((P0, s) # xs) ! length xs) = LanguageCon.com.Skip ∧
      (∃ ys. (n, Γ, (P1, snd (((P0, s) # xs) ! length xs)) # ys) ∈ cptn-mod-nest-call
∧
          (Q, t) # cfg1 =
          map (λ(P, s). (LanguageCon.com.Seq P P1, s)) xs @
          (P1, snd (((P0, s) # xs) ! length xs)) # ys)
    show ?thesis
    proof (cases xs)
      case Nil thus ?thesis using cfg a0 ass by auto
    next
      case (Cons xa xsa)
      then obtain a b where xa:xa = (a,b) by fastforce
      obtain pps :: (('a, 'b, 'c, 'd) LanguageCon.com × ('a, 'c) xstate) list where
          (Q, t) # cfg1 = ((case (a, b) of (c, x) ⇒ (LanguageCon.com.Seq c P1,
x)) # map (λ(c, y).
                        (LanguageCon.com.Seq c P1, y)) xsa) @
                        (P1, snd (((P0, s) # xs) ! length xs)) # pps
        using xa ass local.Cons by moura
       then show ?thesis
         by (simp add: xa local.Cons)
    qed
  next
    assume ass:fst (((P0, s) # xs) ! length xs) = LanguageCon.com.Throw ∧
      snd (last ((P0, s) # xs)) = Normal s' ∧
      s = Normal s'' ∧
    (∃ ys. (n, Γ, (LanguageCon.com.Throw, Normal s') # ys) ∈ cptn-mod-nest-call
∧
          (Q, t) # cfg1 =
          map (λ(P, s). (LanguageCon.com.Seq P P1, s)) xs @
          (LanguageCon.com.Throw, Normal s') # ys)
    thus ?thesis
    proof (cases xs)
      case Nil thus ?thesis using cfg a0 ass by auto
    next
      case (Cons xa xsa)
      then obtain a b where xa:xa = (a,b) by fastforce
      obtain pps :: (('a, 'b, 'c, 'd) LanguageCon.com × ('a, 'c) xstate) list where
       (Q, t) # cfg1 = ((case (a, b) of (c, x) ⇒ (LanguageCon.com.Seq c P1, x))
# map (λ(c, y).
```

$(LanguageCon.com.Seq\ c\ P1,\ y))\ xsa)\ @\ (LanguageCon.com.Throw,$
$Normal\ s')\ \#\ pps$
using *ass local.Cons xa* **by** *force*
then show *?thesis*
by (*simp add*: *local.Cons xa*)
qed
qed
qed

lemma *catch-xs-no-empty*: **assumes**
seq:$catch\text{-}cond\text{-}nest\ ((Q,t)\#cfg1)\ P1\ xs\ P0\ s\ s''\ s'\ \Gamma\ n$ **and**
cfg:$cfg = (LanguageCon.com.Catch\ P0\ P1,\ s)\ \#\ (Q,\ t)\ \#\ cfg1$ **and**
a0:$SmallStepCon.redex\ (LanguageCon.com.Catch\ P0\ P1) = LanguageCon.com.Call$
f
shows$\exists\ Q'\ xs'.\ Q{=}Catch\ Q'\ P1 \wedge xs{=}(Q',t)\#xs'$
using *seq*
unfolding *lift-catch-def catch-cond-nest-def*
proof
assume $(Q,\ t)\ \#\ cfg1 = map\ (\lambda(P,\ s).\ (LanguageCon.com.Catch\ P\ P1,\ s))$
xs
thus *?thesis* **by** *auto*
next
assume $fst\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P0,\ s)\ \#\ xs)) = Normal\ s' \wedge$
$s = Normal\ s'' \wedge$
$(\exists\ ys.\ (n,\ \Gamma,\ (P1,\ snd\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call$
$\wedge$
$(Q,\ t)\ \#\ cfg1 = map\ (\lambda(P,\ s).\ (LanguageCon.com.Catch\ P\ P1,\ s))\ xs\ @$
$(P1,\ snd\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \vee$
$fst\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Skip \wedge$
$(\exists\ ys.\ (n,\ \Gamma,\ (LanguageCon.com.Skip,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$(Q,\ t)\ \#\ cfg1 =$
$map\ (\lambda(P,\ s).\ (LanguageCon.com.Catch\ P\ P1,\ s))\ xs\ @$
$(LanguageCon.com.Skip,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys)$
thus *?thesis*
proof
assume *ass*:$fst\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs) = LanguageCon.com.Throw \wedge$
$snd\ (last\ ((P0,\ s)\ \#\ xs)) = Normal\ s' \wedge$
$s = Normal\ s'' \wedge$
$(\exists\ ys.\ (n,\ \Gamma,\ (P1,\ snd\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys) \in$
$cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$(Q,\ t)\ \#\ cfg1 = map\ (\lambda(P,\ s).\ (LanguageCon.com.Catch\ P\ P1,\ s))$
$xs\ @$
$(P1,\ snd\ (((P0,\ s)\ \#\ xs)\ !\ length\ xs))\ \#\ ys)$
show *?thesis*
proof (*cases xs*)
case *Nil* **thus** *?thesis* **using** *cfg a0 ass* **by** *auto*
next

case (*Cons xa xsa*)
then obtain *a b* **where** *xa*:*xa* = (*a*,*b*) **by** *fastforce*
obtain *pps* :: (('*a*, '*b*, '*c*, '*d*) *LanguageCon.com* × ('*a*, '*c*) *xstate*) *list* **where**
(*Q*, *t*) # *cfg1* = ((*case* (*a*, *b*) *of* (*c*, *x*) ⇒ (*LanguageCon.com.Catch c P1*, *x*)) #
map (λ(*c*, *y*). (*LanguageCon.com.Catch c P1*, *y*)) *xsa*) @
(*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *pps*
using *ass local.Cons xa* **by** *moura*
then show *?thesis*
by (*simp add*: *local.Cons xa*)
qed
next
assume *ass*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*, Γ, (*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*Q*, *t*) # *cfg1* =
map (λ(*P*, *s*). (*LanguageCon.com.Catch P P1*, *s*)) *xs* @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*)
thus *?thesis*
proof (*cases xs*)
case *Nil* **thus** *?thesis* **using** *cfg a0 ass* **by** *auto*
next
case (*Cons xa xsa*)
then obtain *a b* **where** *xa*:*xa* = (*a*,*b*) **by** *fastforce*
obtain *pps* :: (('*a*, '*b*, '*c*, '*d*) *LanguageCon.com* × ('*a*, '*c*) *xstate*) *list* **where**
(*Q*, *t*) # *cfg1* = ((*case* (*a*, *b*) *of* (*c*, *x*) ⇒
(*LanguageCon.com.Catch c P1*, *x*)) # *map* (λ(*c*, *y*).
(*LanguageCon.com.Catch c P1*, *y*)) *xsa*) @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *pps*
using *ass local.Cons xa* **by** *force*
then show *?thesis*
by (*simp add*: *local.Cons xa*)
qed
qed
qed

lemma *redex-call-cptn-mod-min-nest-call-gr-zero*:
assumes *a0*:*min-call n* Γ *cfg* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1* **and**
a2:*redex P* = *Call f* ∧
Γ *f* = *Some bdy* ∧ (∃ *sa*. *s*=*Normal sa*) ∧ *t*=*s* **and**
a3:$\Gamma\vdash_c$(*P*,*s*)→(*Q*,*t*)
shows *n*>*0*
using *a0 a1 a2 a3*
proof (*induct P arbitrary*: *Q cfg1 cfg s t n*)
case (*Call f1*) **thus** *?case*
by (*metis SmallStepCon.redex.simps*(*7*) *elim-cptn-mod-nest-call-n-greater-zero min-call-def option.distinct*(*1*) *stepc-Normal-elim-cases*(*9*))
next

case (*Seq P0 P1*)
then obtain *xs s′ s″* **where**
p0-cptn:(*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
seq:*seq-cond-nest* ((*Q*,*t*)#*cfg1*) *P1 xs P0 s s″ s′* Γ *n*
using *div-seq-nest*[*of n* Γ *cfg*] **unfolding** *min-call-def* **by** *blast*
then obtain *m* **where** *min*:*min-call m* Γ ((*P0*, *s*)#*xs*)
using *minimum-nest-call* **by** *blast*
have *xs′*:∃ *Q′ xs′*. *Q*=*Seq Q′ P1* ∧ *xs*=(*Q′*,*t*)#*xs′*
using *seq Seq seq-xs-no-empty* **by** *auto*
then have *0*<*m* **using** *Seq*(*1*,*5*,*6*) *min*
using *SmallStepCon.redex.simps*(*4*) *stepc-elim-cases-Seq-Seq* **by** *fastforce*
thus *?case* **by** (*metis min min-call-def not-gr0 p0-cptn*)
next
case (*Catch P0 P1*)
then obtain *xs s′ s″* **where**
p0-cptn:(*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
seq:*catch-cond-nest* ((*Q*,*t*)#*cfg1*) *P1 xs P0 s s″ s′* Γ *n*
using *div-catch-nest*[*of n* Γ *cfg*] **unfolding** *min-call-def* **by** *blast*
then obtain *m* **where** *min*:*min-call m* Γ ((*P0*, *s*)#*xs*)
using *minimum-nest-call* **by** *blast*
obtain *Q′ xs′* **where** *xs′*:*Q*=*Catch Q′ P1* ∧ *xs*=(*Q′*,*t*)#*xs′*
using *catch-xs-no-empty*[*OF seq Catch*(*4*)] *Catch* **by** *blast*
then have *0*<*m* **using** *Catch*(*1*,*5*,*6*) *min*
using *SmallStepCon.redex.simps*(*4*) *stepc-elim-cases-Catch-Catch* **by** *fastforce*
thus *?case* **by** (*metis min min-call-def not-gr0 p0-cptn*)
qed(*auto*)

lemma *elim-redex-call-cptn-mod-min-nest-call*:
assumes *a0*:*min-call n* Γ *cfg* **and**
a1:*cfg* = (*P*,*s*)#(*Q*,*t*)#*cfg1* **and**
a2:*redex P* = *Call f* ∧
Γ *f* = *Some bdy* ∧ (∃ *sa*. *s*=*Normal sa*) ∧ *t*=*s* **and**
a3:*biggest-nest-call P s* ((*Q*,*t*)#*cfg1*) Γ *n*
shows *min-call n* Γ ((*Q*,*t*)#*cfg1*)
using *a0 a1 a2 a3*
proof (*induct P arbitrary*: *Q cfg1 cfg s t n*)
case *Cond* **thus** *?case* **by** *fastforce*
next
case (*Seq P0 P1*)
then obtain *xs s′ s″* **where**
p0-cptn:(*n*, Γ,(*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* **and**
seq:*seq-cond-nest* ((*Q*,*t*)#*cfg1*) *P1 xs P0 s s″ s′* Γ *n*
using *div-seq-nest*[*of n* Γ *cfg*] **unfolding** *min-call-def* **by** *blast*

show *?case* **using** *seq* **unfolding** *seq-cond-nest-def*
proof
assume *ass*:(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*

then obtain Q' xs' **where** xs':Q=*Seq* Q' *P1* ∧ xs=(Q',t)#xs'
unfolding *lift-def* **by** *fastforce*
then have *ctpn-P0*:(*P0*, *s*) # *xs* = (*P0*, *s*) # (Q', *t*) # xs' **by** *auto*
then have *min-p0*:*min-call n* Γ ((*P0*, *s*)#*xs*)
using *min-call-seq-c1-not-finish*[*OF Seq*(*3*) *Seq*(*4*) *p0-cptn*] *ass* **by** *auto*
then have *ex-xs*:∃ *xs*. *min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*
using *ass* **by** *auto*
then have *min-xs*:*min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*
using *min-p0 ass* **by** *auto*
have *xs*= (*SOME xs*. (*min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*))
proof −
have ∀ *xsa*. *min-call n* Γ ((*P0*, *s*)#*xsa*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xsa* ⟶ *xsa* = *xs*
using xs' *ass* **by** (*metis map-lift-eq-xs-xs'*)
thus *?thesis* **using** *min-xs some-equality* **by** (*metis* (*mono-tags*, *lifting*))
qed
then have *big*:*biggest-nest-call P0 s* ((Q', *t*) # xs') Γ *n*
using *biggest-nest-call.simps*(*1*)[*of P0 P1 s* ((*Q*, *t*) # *cfg1*) Γ *n*]
Seq(*6*) xs' *ex-xs* **by** *auto*
have *reP0*:*redex P0* = (*Call f*) ∧ Γ *f* = *Some bdy* ∧
(∃ *saa*. *s* = *Normal saa*) ∧ *t* = *s* **using** *Seq*(*5*) xs' **by** *auto*
have *min-call*:*min-call n* Γ ((Q', *t*) # xs')
using *Seq*(*1*)[*OF min-p0 ctpn-P0 reP0*] *big* xs' *ass* **by** *auto*
thus *?thesis* **using** *min-call-seq-not-finish*[*OF min-call*] *ass* xs' **by** *blast*
next
assume *ass*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*, Γ, (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs* @ (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) ∨
fst (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* ∧
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal* s' ∧
s = *Normal* s'' ∧
(∃ *ys*. (*n*, Γ, (*LanguageCon.com.Throw*, *Normal* s') # *ys*) ∈ *cptn-mod-nest-call* ∧
(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs* @ (*LanguageCon.com.Throw*, *Normal* s') # *ys*)
{**assume** *ass*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(∃ *ys*. (*n*, Γ, (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs* @ (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
have *?thesis*
proof (*cases xs*)
case *Nil* **thus** *?thesis* **using** *Seq ass* **by** *fastforce*
next

case (*Cons xa xsa*)
then obtain *ys* **where**
seq2-ass:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Skip* ∧
(*n*, Γ, (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) ∈ *cptn-mod-nest-call* ∧
(*Q*, *t*) # *cfg1* = *map* (*lift P1*) (*xa*#*xsa*) @ (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*
using *ass* **by** *auto*
then obtain *mq mp1* **where**
min-call-q:*min-call mq* Γ ((*P0*, *s*) # *xs*) **and**
min-call-p1:*min-call mp1* Γ ((*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)

using *seq2-ass minimum-nest-call p0-cptn* **by** *fastforce*
then have *mp*: *mq*≤*n* ∧ *mp1* ≤*n*
using *seq2-ass min-call-less-eq-n*[*of n* Γ *P0 s xs P1 ys mq mp1*]
Seq(*3*,*4*) *p0-cptn* **by** (*simp add*: *last-length*)
have *min-call*:*min-call n* Γ ((*P0*, *s*) # *xs*) ∨
min-call n Γ ((*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
using *seq2-ass min-call-seq2*[*of n* Γ *P0 P1 s* (*Q*, *t*) # *cfg1 xs ys*]
Seq(*3*,*4*) *p0-cptn* **by** (*simp add*: *last-length local.Cons*)
from *seq2-ass* **obtain** *Q′* **where** *Q′*:*Q*=*Seq Q′ P1* ∧ *xa*=(*Q′*,*t*)
unfolding *lift-def*
by (*metis* (*mono-tags*, *lifting*) *fst-conv length-greater-0-conv*
list.simps(*3*) *list.simps*(*9*) *nth-Cons-0 nth-append prod.case-eq-if prod.collapse snd-conv*)
then have *q′-n-cptn*:(*n*,Γ,(*Q′*,*t*)#*xsa*)∈*cptn-mod-nest-call* **using** *p0-cptn Q′ Cons*
using *elim-cptn-mod-nest-call-n* **by** *blast*
show *?thesis*
proof(*cases mp1*=*n*)
case *True*
then have *min-call n* Γ ((*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
using *min-call-p1* **by** *auto*
then have *min-P1*:*min-call n* Γ ((*P1*, *snd* ((*xa* # *xsa*) ! *length xsa*)) # *ys*)
using *Cons seq2-ass* **by** *fastforce*
then have *p1-n-cptn*:(*n*, Γ, (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
using *Seq.prems*(*1*) *Seq.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def* **by** *blast*
also then have (∀ *m*<*n*. (*m*, Γ, (*Q*, *t*) # *cfg1*) ∉ *cptn-mod-nest-call*)
proof−
{ **fix** *m*
assume *ass*:*m*<*n*
{ **assume** *Q-m*:(*m*, Γ, (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
then have *False* **using** *min-P1 ass Q′ Cons* **unfolding** *min-call-def*
proof −
assume *a1*: (*n*, Γ, (*P1*, *snd* ((*xa* # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call* ∧ (∀ *m*<*n*. (*m*, Γ, (*P1*, *snd* ((*xa* # *xsa*) ! *length xsa*)) # *ys*) ∉ *cptn-mod-nest-call*)
have *f2*: ∀ *n f ps*. (*n*, *f*, *ps*) ∉ *cptn-mod-nest-call* ∨ (∀ *x c ca psa*. *ps* ≠

(*LanguageCon.com.Seq* (*c*::(′*b*, ′*a*, ′*c*,′*d*) *LanguageCon.com*) *ca*, *x*) # *psa* ∨ (∃ *ps* *b ba*. (*n*, *f*, (*c*, *x*) # *ps*) ∈ *cptn-mod-nest-call* ∧ *seq-cond-nest psa ca ps c x ba b f n*))
using *div-seq-nest* **by** *blast*
have *f3*: (*P1*, *snd* (*last* ((*Q*′, *t*) # *xsa*))) # *ys* = (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*
by (*simp add*: *Q*′ *last-length local.Cons*)
have *fst* (*last* ((*Q*′, *t*) # *xsa*)) = *LanguageCon.com.Skip*
by (*metis* (*no-types*) *Q*′ *last-ConsR last-length list.distinct*(*1*) *local.Cons seq2-ass*)
then show *?thesis*
using *f3 f2 a1* **by** (*metis* (*no-types*) *Cons-lift-append Q*′ *Seq-P-Ends-Normal Q-m ass seq2-ass*)
qed
}
} **then show** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
next
case *False*
then have *mp1*<*n* **using** *mp* **by** *auto*
then have *not-min-call-p1-n*:¬ *min-call n* Γ ((*P1*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*)
using *min-call-p1 last-length* **unfolding** *min-call-def* **by** *metis*
then have *min-call*:*min-call n* Γ ((*P0*, *s*) # *xs*)
using *min-call last-length* **unfolding** *min-call-def* **by** *metis*
then have (*P0*, *s*) # *xs* = (*P0*, *s*) # *xa*#*xsa*
using *Cons* **by** *auto*
then have *big*:*biggest-nest-call P0 s* (((*Q*′,*t*))#*xsa*) Γ *n*
proof−
have ¬(∃ *xs*. *min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*)
using *min-call seq2-ass Cons*
proof −
have *min-call n* Γ ((*LanguageCon.com.Seq P0 P1*, *s*) # (*Q*, *t*) # *cfg1*)
using *Seq.prems*(*1*) *Seq.prems*(*2*) **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *Seq-P-Not-finish append-Nil2 list.simps*(*3*) *local.Cons min-call-def same-append-eq seq seq2-ass*)
qed
moreover have ∃ *xs ys*. *cond-seq-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*
using *seq2-ass p0-cptn* **unfolding** *cond-seq-1-def*
by (*metis last-length local.Cons*)
moreover have (*SOME xs*. ∃ *ys*. *cond-seq-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*) = *xs*
proof −
let *?P* = λ*xsa*. ∃ *ys*. (*n*, Γ, (*P0*, *s*) # *xsa*) ∈ *cptn-mod-nest-call* ∧
fst (*last* ((*P0*, *s*) # *xsa*)) = *LanguageCon.com.Skip* ∧
(*n*, Γ, (*P1*, *snd* (*last* ((*P0*, *s*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call*

$\wedge$
$(Q,\ t)\ \#\ cfg1\ =\ map\ (lift\ P1)\ xsa\ @\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xsa)))\ \#\ ys$
have $(\bigwedge x.\ \exists ys.\ (n,\ \Gamma,\ (P0,\ s)\ \#\ x) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$fst\ (last\ ((P0,\ s)\ \#\ x)) = LanguageCon.com.Skip\ \wedge$
$(n,\ \Gamma,\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ x)))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$(Q,\ t)\ \#\ cfg1\ =\ map\ (lift\ P1)\ x\ @\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ x)))\ \#\ ys \Longrightarrow$
$x = xs)$
by (*metis Seq-P-Ends-Normal cptn-mod-nest-call.CptnModNestSeq2 seq*)
moreover have $\exists ys.\ (n,\ \Gamma,\ (P0,\ s)\ \#\ xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$fst\ (last\ ((P0,\ s)\ \#\ xs)) = LanguageCon.com.Skip\ \wedge$
$(n,\ \Gamma,\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$(Q,\ t)\ \#\ cfg1\ =\ map\ (lift\ P1)\ xs\ @\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys$
using *ass p0-cptn* **by** (*simp add*: *last-length*)
ultimately show *?thesis* **using** *some-equality*[*of ?P xs*]
unfolding *cond-seq-1-def* **by** *blast*
qed
moreover have $(SOME\ ys.\ cond\text{-}seq\text{-}1\ n\ \Gamma\ P0\ s\ xs\ P1\ ((Q,\ t)\ \#\ cfg1)\ ys) = ys$
proof –
let $?P = \lambda ys.\ (n,\ \Gamma,\ (P0,\ s)\ \#\ xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$fst\ (last\ ((P0,\ s)\ \#\ xs)) = LanguageCon.com.Skip\ \wedge$
$(n,\ \Gamma,\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$(Q,\ t)\ \#\ cfg1\ =\ map\ (lift\ P1)\ xs\ @\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys$
have $(n,\ \Gamma,\ (P0,\ s)\ \#\ xs) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$fst\ (last\ ((P0,\ s)\ \#\ xs)) = LanguageCon.com.Skip\ \wedge$
$(n,\ \Gamma,\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call\ \wedge$
$(Q,\ t)\ \#\ cfg1\ =\ map\ (lift\ P1)\ xs\ @\ (P1,\ snd\ (last\ ((P0,\ s)\ \#\ xs)))\ \#\ ys$
using *p0-cptn seq2-ass Cons* **by** (*simp add*: *last-length*)
then show *?thesis* **using** *some-equality*[*of ?P ys*]
unfolding *cond-seq-1-def* **by** *fastforce*
qed
ultimately have *biggest-nest-call P0 s xs* Γ *n*
using *not-min-call-p1-n Seq*(*6*)
biggest-nest-call.simps(*1*)[*of P0 P1 s* $(Q,\ t)\ \#\ cfg1\ \Gamma\ n$]
by *presburger*
then show *?thesis* **using** *Cons* Q' **by** *auto*
qed
have $C{:}(P0,\ s)\ \#\ xs = (P0,\ s)\ \#\ (Q',\ t)\ \#\ xsa$ **using** *Cons* Q' **by** *auto*
have *reP0*:$redex\ P0 = (Call\ f) \wedge \Gamma\ f = Some\ bdy\ \wedge$
$(\exists saa.\ s = Normal\ saa) \wedge t = s$ **using** *Seq*(*5*) Q' **by** *auto*
then have *min-call*:$min\text{-}call\ n\ \Gamma\ ((Q',\ t)\ \#\ xsa)$ **using** *Seq*(*1*)[*OF min-call C reP0 big*]
by *auto*

have *p1-n-cptn*:(n, Γ, (Q, t) # *cfg1*) $\in$ *cptn-mod-nest-call*
using *Seq.prems*(*1*) *Seq.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def*
by *blast*
also then have ($\forall m<n.$ (m, Γ, (Q, t) # *cfg1*) $\notin$ *cptn-mod-nest-call*)
proof−
{ **fix** m
assume *ass*:$m<n$
{ **assume** *Q-m*:(m, Γ, (Q, t) # *cfg1*) $\in$ *cptn-mod-nest-call*
then obtain *xsa′ s1 s1′* **where**
p0-cptn:(m, Γ,(Q', t)#*xsa′*) $\in$ *cptn-mod-nest-call* **and**
seq:*seq-cond-nest cfg1 P1 xsa′ Q′ t s1 s1′* Γ m
using *div-seq-nest*[*of m* Γ (Q, t) # *cfg1*] Q' **by** *blast*
then have *xsa*=*xsa′*
using *seq2-ass*
Seq-P-Ends-Normal[*of cfg1 P1 xsa Q′ t ys m* Γ *xsa′ s1 s1′*] *Cons*
by (*metis Cons-lift-append Q′ Q-m last.simps last-length list.inject list.simps*(*3*))
then have *False* **using** *min-call p0-cptn ass* **unfolding** *min-call-def*
by *auto*
}
} **then show** *?thesis* **by** *auto* **qed**

ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed
qed
}**note** *l*=*this*
{**assume** *ass*:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal s′* $\wedge$
s = *Normal s″* $\wedge$ ($\exists$ *ys*. (n, Γ, (*LanguageCon.com.Throw*, *Normal s′*) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(Q, t) # *cfg1* = *map* (*lift P1*) *xs* @ (*LanguageCon.com.Throw*, *Normal s′*) # *ys*)
have *?thesis*
proof (*cases* $\Gamma\vdash_c$(*LanguageCon.com.Seq P0 P1*, *s*) $\rightarrow$ (Q,t))
case *True*
thus *?thesis*
proof (*cases xs*)
case *Nil* **thus** *?thesis* **using** *Seq ass* **by** *fastforce*
next
case (*Cons xa xsa*)
then obtain *ys* **where**
seq2-ass:*fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal s′* $\wedge$
s = *Normal s″* $\wedge$ (n, Γ, (*LanguageCon.com.Throw*, *Normal s′*) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(Q, t) # *cfg1* = *map* (*lift P1*) *xs* @ (*LanguageCon.com.Throw*, *Normal s′*) # *ys*
using *ass* **by** *auto*
then have *t-eq*:*t*=*Normal s″* **using** *Seq* **by** *fastforce*

obtain *mq mp1* **where**
min-call-q:*min-call mq* Γ ((*P0*, *s*) # *xs*) **and**
min-call-p1:*min-call mp1* Γ ((*Throw*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
using *seq2-ass minimum-nest-call p0-cptn* **by** (*metis last-length*)
then have *mp1-zero*:*mp1*=*0* **by** (*simp add*: *throw-min-nested-call-0*)
then have *min-call*: *min-call n* Γ ((*P0*, *s*) # *xs*)
using *seq2-ass min-call-seq3*[*of n* Γ *P0 P1 s* (*Q*, *t*) # *cfg1 s''* *xs s'* *ys*]
Seq(*3*,*4*) *p0-cptn* **by** (*metis last-length*)
have *n-z*:*n*>*0* **using** *redex-call-cptn-mod-min-nest-call-gr-zero*[*OF Seq*(*3*) *Seq*(*4*) *Seq*(*5*) *True*]
by *auto*
from *seq2-ass* **obtain** *Q'* **where** *Q'*:*Q*=*Seq Q' P1* ∧ *xa*=(*Q'*,*t*)
unfolding *lift-def* **using** *Cons*
proof −
assume *a1*: ⋀*Q'*. *Q* = *LanguageCon.com.Seq Q' P1* ∧ *xa* = (*Q'*, *t*) ⟹ *thesis*
have (*LanguageCon.com.Seq* (*fst xa*) *P1*, *snd xa*) = ((*Q*, *t*) # *cfg1*) ! *0*
using *seq2-ass* **unfolding** *lift-def*
by (*simp add*: *Cons case-prod-unfold*)
then show *?thesis*
using *a1* **by** *fastforce*
qed
have *big-call*:*biggest-nest-call P0 s* ((*Q'*,*t*)#*xsa*) Γ *n*
proof−
have ¬(∃*xs*. *min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift P1*) *xs*)
using *min-call seq2-ass Cons Seq.prems*(*1*) *Seq.prems*(*2*)
by (*metis Seq-P-Not-finish append-Nil2 list.simps*(*3*) *min-call-def same-append-eq seq*)
moreover have ¬(∃*xs ys*. *cond-seq-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*)
using *min-call seq2-ass p0-cptn Cons Seq.prems*(*1*) *Seq.prems*(*2*)
unfolding *cond-seq-1-def*
by (*metis com.distinct*(*17*) *com.distinct*(*71*) *last-length*
map-lift-some-eq seq-and-if-not-eq(*4*))
moreover have (*SOME xs*. ∃*ys s' s''*. *cond-seq-2 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys s' s''*) = *xs*
proof−
let *?P*=λ*xsa*. ∃*ys s' s''*. *s*= *Normal s''* ∧
(*n*,Γ, (*P0*, *s*)#*xs*) ∈ *cptn-mod-nest-call* ∧
fst(*last* ((*P0*, *s*)#*xs*)) = *Throw* ∧
snd(*last* ((*P0*, *s*)#*xs*)) = *Normal s'* ∧
(*n*,Γ,(*Throw*,*Normal s'*)#*ys*) ∈ *cptn-mod-nest-call* ∧
((*Q*, *t*) # *cfg1*)=(*map* (*lift P1*) *xs*)@((*Throw*,*Normal s'*)#*ys*)
have (⋀*x*. ∃*ys s' s''*. *s*= *Normal s''* ∧
(*n*,Γ, (*P0*, *s*)#*x*) ∈ *cptn-mod-nest-call* ∧
fst(*last* ((*P0*, *s*)#*x*)) = *Throw* ∧
snd(*last* ((*P0*, *s*)#*x*)) = *Normal s'* ∧

$(n,\Gamma,(Throw,Normal\ s')\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$((Q,\ t)\ \#\ cfg1)=(map\ (lift\ P1)\ x)@((Throw,Normal\ s')\#ys) \Longrightarrow$
$x=xs$) **using** *map-lift-some-eq seq2-ass* **by** *fastforce*
moreover have $\exists\, ys\ s'\ s''.\ s=\ Normal\ s'' \wedge$
$(n,\Gamma,\ (P0,\ s)\#xs) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$fst(last\ ((P0,\ s)\#xs)) = Throw \wedge$
$snd(last\ ((P0,\ s)\#xs)) = Normal\ s' \wedge$
$(n,\Gamma,(Throw,Normal\ s')\#ys) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$((Q,\ t)\ \#\ cfg1)=(map\ (lift\ P1)\ xs)@((Throw,Normal\ s')\#ys)$
using *ass p0-cptn* **by** (*simp add*: *last-length Cons*)
ultimately show *?thesis* **using** *some-equality*[*of ?P xs*]
unfolding *cond-seq-2-def* **by** *blast*
qed
ultimately have *biggest-nest-call P0 s xs* Γ *n*
using *Seq*(*6*)
biggest-nest-call.simps(*1*)[*of P0 P1 s* $(Q,\ t)\ \#\ cfg1\ \Gamma\ n$]
by *presburger*
then show *?thesis* **using** *Cons Q'* **by** *auto*
qed
have *min-call*:*min-call n* Γ $((Q',t)\#xsa)$
using *Seq*(*1*)[*OF min-call - - big-call*] *Seq*(*5*) *Cons Q'* **by** *fastforce*
then have *p1-n-cptn*:$(n,\ \Gamma,\ (Q,\ t)\ \#\ cfg1) \in cptn\text{-}mod\text{-}nest\text{-}call$
using *Seq.prems*(*1*) *Seq.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def*
by *blast*
also then have $(\forall\, m<n.\ (m,\ \Gamma,\ (Q,\ t)\ \#\ cfg1) \notin cptn\text{-}mod\text{-}nest\text{-}call)$
proof−
{ **fix** *m*
assume *ass*:$m<n$
{ **assume** *Q-m*:$(m,\ \Gamma,\ (Q,\ t)\ \#\ cfg1) \in cptn\text{-}mod\text{-}nest\text{-}call$
then obtain *xsa' s1 s1'* **where**
p0-cptn:$(m,\ \Gamma,(Q',\ t)\#xsa') \in cptn\text{-}mod\text{-}nest\text{-}call$ **and**
seq:*seq-cond-nest cfg1 P1 xsa' Q'* (*Normal s''*) *s1 s1'* Γ *m*
using *div-seq-nest*[*of m* Γ $(Q,\ t)\ \#\ cfg1$] *Q' t-eq* **by** *blast*
then have $xsa=xsa'$
using *seq2-ass*
Seq-P-Ends-Abort[*of cfg1 P1 xsa s' ys Q' s'' m* Γ *xsa' s1 s1'*] *Cons Q' Q-m*
by (*simp add*: *Cons-lift-append last-length t-eq*)
then have *False* **using** *min-call p0-cptn ass* **unfolding** *min-call-def* **by** *auto*
}
} **then show** *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed
next
case *False*
then have *env*:$\Gamma\vdash_c(LanguageCon.com.Seq\ P0\ P1,\ s) \rightarrow_e (Q,t)$ **using** *Seq*
by (*meson elim-cptn-mod-nest-step-c min-call-def*)
moreover then have *Q*:$Q=Seq\ P0\ P1$ **using** *env-c-c'* **by** *blast*

ultimately show *?thesis* **using** *Seq*
proof –
obtain *nn* :: $(('b, 'a, 'c,'d)$ *LanguageCon.com* $\times$ $('b, 'c)$ *xstate*$)$ *list* $\Rightarrow$ $('a \Rightarrow ('b, 'a, 'c,'d)$ *LanguageCon.com option*$)$ $\Rightarrow$ *nat* $\Rightarrow$ *nat*
where
f1: $\forall$ *x0 x1 x2*. $(\exists$ *v3*<*x2*. $($*v3*, *x1*, *x0*$)$ $\in$ *cptn-mod-nest-call*$)$ = $($*nn x0 x1 x2* < *x2* $\wedge$ $($*nn x0 x1 x2*, *x1*, *x0*$)$ $\in$ *cptn-mod-nest-call*$)$
by *moura*
have *f2*: $(n, \Gamma,$ $($*LanguageCon.com.Seq P0 P1*, *s*$)$ # (Q, t) # *cfg1*$)$ $\in$ *cptn-mod-nest-call* $\wedge$ $(\forall n. \neg n < n \vee (n, \Gamma,$ $($*LanguageCon.com.Seq P0 P1*, *s*$)$ # (Q, t) # *cfg1*$)$ $\notin$ *cptn-mod-nest-call*$)$
using *local.Seq*(3) *local.Seq*(4) *min-call-def* **by** *blast*
then have $\neg$ *nn* $((Q, t)$ # *cfg1*$)$ Γ $n < n$ $\vee$ $($*nn* $((Q, t)$ # *cfg1*$)$ Γ n, Γ, (Q, t) # *cfg1*$)$ $\notin$ *cptn-mod-nest-call*
using *False env env-c-c′ not-func-redex-cptn-mod-nest-n-env*
by $($*metis Seq.prems*(1) *Seq.prems*(2) *min-call-def*$)$
then show *?thesis*
using *f2 f1* **by** $($*meson elim-cptn-mod-nest-call-n min-call-def*$)$
qed
qed
}
thus *?thesis* **using** *l ass* **by** *fastforce*
qed
next
case $($*Catch P0 P1*$)$
then obtain *xs s′ s′′* **where**
p0-cptn:$(n, \Gamma,$($P0$, *s*)#*xs*$)$ $\in$ *cptn-mod-nest-call* **and**
catch:*catch-cond-nest* $(($Q$,t$)#*cfg1*$)$ *P1 xs P0 s s′′ s′* Γ *n*
using *div-catch-nest*[*of n* Γ *cfg*] **unfolding** *min-call-def* **by** *blast*

show *?case* **using** *catch* **unfolding** *catch-cond-nest-def*
proof
assume *ass*:(Q, t) # *cfg1* = *map* $($*lift-catch P1*$)$ *xs*
then obtain *Q′ xs′* **where** *xs′*:*Q*=*Catch Q′ P1* $\wedge$ *xs*=$(Q′,t)$#*xs′*
unfolding *lift-catch-def* **by** *fastforce*
then have *ctpn-P0*:$(P0, s)$ # *xs* = $(P0, s)$ # $(Q′, t)$ # *xs′* **by** *auto*
then have *min-p0*:*min-call n* Γ $(($*P0*, *s*$)$#*xs*$)$
using *min-call-catch-c1-not-finish*[*OF Catch*(3) *Catch*(4) *p0-cptn*] *ass* **by** *auto*
then have *ex-xs*:$\exists$ *xs*. *min-call n* Γ $(($*P0*, *s*$)$#*xs*$)$ $\wedge$ (Q, t) # *cfg1* = *map* $($*lift-catch P1*$)$ *xs*
using *ass* **by** *auto*
then have *min-xs*:*min-call n* Γ $(($*P0*, *s*$)$#*xs*$)$ $\wedge$ (Q, t) # *cfg1* = *map* $($*lift-catch P1*$)$ *xs*
using *min-p0 ass* **by** *auto*
have *xs*= $($*SOME xs*. $($*min-call n* Γ $(($*P0*, *s*$)$#*xs*$)$ $\wedge$ (Q, t) # *cfg1* = *map* $($*lift-catch P1*$)$ *xs*$))$
proof –
have $\forall$ *xsa*. *min-call n* Γ $(($*P0*, *s*$)$#*xsa*$)$ $\wedge$ (Q, t) # *cfg1* = *map* $($*lift-catch*

$P1$) $xsa \longrightarrow xsa = xs$
using xs' ass **by** ($metis$ map-$lift$-$catch$-eq-xs-xs')
thus $?thesis$ **using** min-xs $some$-$equality$ **by** ($metis$ ($mono$-$tags$, $lifting$))
qed
then have big:$biggest$-$nest$-$call$ $P0$ s ((Q', t) # xs') Γ n
using $biggest$-$nest$-$call.simps$(2)[of $P0$ $P1$ s ((Q, t) # $cfg1$) Γ n]
$Catch$(6) xs' ex-xs **by** $auto$
have $reP0$:$redex$ $P0$ = ($Call$ f) $\wedge$ Γ f = $Some$ bdy $\wedge$
($\exists$ saa. s = $Normal$ saa) $\wedge$ t = s **using** $Catch$(5) xs' **by** $auto$
have min-$call$:min-$call$ n Γ ((Q', t) # xs')
using $Catch$(1)[OF min-$p0$ $ctpn$-$P0$ $reP0$] big xs' ass **by** $auto$
thus $?thesis$ **using** min-$call$-$catch$-not-$finish$[OF min-$call$] ass xs' **by** $blast$
next
assume ass:fst ((($P0$, s) # xs) ! $length$ xs) = $LanguageCon.com.Throw$ $\wedge$
snd ($last$ (($P0$, s) # xs)) = $Normal$ s' $\wedge$
s = $Normal$ s'' $\wedge$
($\exists$ ys. (n, Γ, ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys) $\in$ $cptn$-mod-$nest$-$call$ $\wedge$
(Q, t) # $cfg1$ = map ($lift$-$catch$ $P1$) xs @ ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys) $\vee$
fst ((($P0$, s) # xs) ! $length$ xs) = $LanguageCon.com.Skip$ $\wedge$
($\exists$ ys. (n, Γ, ($LanguageCon.com.Skip$, snd ($last$ (($P0$, s) # xs))) # ys) $\in$ $cptn$-mod-$nest$-$call$ $\wedge$
(Q, t) # $cfg1$ = map ($lift$-$catch$ $P1$) xs @ ($LanguageCon.com.Skip$, snd ($last$ (($P0$, s) # xs))) # ys)
{**assume** ass:fst ((($P0$, s) # xs) ! $length$ xs) = $LanguageCon.com.Throw$ $\wedge$
snd ($last$ (($P0$, s) # xs)) = $Normal$ s' $\wedge$
s = $Normal$ s'' $\wedge$
($\exists$ ys. (n, Γ, ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys) $\in$ $cptn$-mod-$nest$-$call$ $\wedge$
(Q, t) # $cfg1$ = map ($lift$-$catch$ $P1$) xs @ ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys)
have $?thesis$
proof ($cases$ xs)
case Nil **thus** $?thesis$ **using** $Catch$ ass **by** $fastforce$
next
case ($Cons$ xa xsa)
then obtain ys **where**
$catch2$-ass:fst ((($P0$, s) # xs) ! $length$ xs) = $LanguageCon.com.Throw$ $\wedge$
snd ($last$ (($P0$, s) # xs)) = $Normal$ s' $\wedge$
s = $Normal$ s'' $\wedge$
(n, Γ, ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys) $\in$ $cptn$-mod-$nest$-$call$ $\wedge$
(Q, t) # $cfg1$ = map ($lift$-$catch$ $P1$) xs @ ($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys
using ass **by** $auto$
then obtain mq $mp1$ **where**
min-$call$-q:min-$call$ mq Γ (($P0$, s) # xs) **and**
min-$call$-$p1$:min-$call$ $mp1$ Γ (($P1$, snd ((($P0$, s) # xs) ! $length$ xs)) # ys)

using *catch2-ass minimum-nest-call p0-cptn* **by** *fastforce*
then have *mp*: $mq \le n \wedge mp1 \le n$
using *catch2-ass min-call-less-eq-n*
Catch(*3*,*4*) *p0-cptn* **by** (*metis last-length*)
have *min-call*:*min-call n* Γ ((*P0*, *s*) # *xs*) $\vee$
min-call n Γ ((*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
using *catch2-ass min-call-catch3*[*of n* Γ *P0 P1 s''* (*Q*, *t*) # *cfg1 xs s' ys*]
Catch(*3*,*4*) *p0-cptn* **by** (*metis last-length*)
from *catch2-ass* **obtain** *Q'* **where** *Q'*:*Q*=*Catch Q' P1* $\wedge$ *xa*=(*Q'*,*t*)
unfolding *lift-catch-def*
proof −
assume *a1*: $\bigwedge$*Q'*. *Q* = *LanguageCon.com.Catch Q' P1* $\wedge$ *xa* = (*Q'*, *t*) $\Longrightarrow$ *thesis*
assume *fst* (((*P0*, *s*) # *xs*) ! *length xs*) = *LanguageCon.com.Throw* $\wedge$ *snd* (*last* ((*P0*, *s*) # *xs*)) = *Normal s'* $\wedge$ *s* = *Normal s''* $\wedge$ (*n*, Γ, (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$ (*Q*, *t*) # *cfg1* = *map* (λ(*P*, *s*). (*LanguageCon.com.Catch P P1*, *s*)) *xs* @ (*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*
then have (*LanguageCon.com.Catch* (*fst xa*) *P1*, *snd xa*) = ((*Q*, *t*) # *cfg1*) ! *0*
by (*simp add*: *local.Cons prod.case-eq-if*)
then show *?thesis*
using *a1* **by** *force*
qed
then have *q'-n-cptn*:(*n*,Γ,(*Q'*,*t*)#*xsa*)$\in$*cptn-mod-nest-call* **using** *p0-cptn Q' Cons*
using *elim-cptn-mod-nest-call-n* **by** *blast*
show *?thesis*
proof(*cases mp1*=*n*)
case *True*
then have *min-call n* Γ ((*P1*, *snd* (((*P0*, *s*) # *xs*) ! *length xs*)) # *ys*)
using *min-call-p1* **by** *auto*
then have *min-P1*:*min-call n* Γ ((*P1*, *snd* ((*xa* # *xsa*) ! *length xsa*)) # *ys*)
using *Cons catch2-ass* **by** *fastforce*
then have *p1-n-cptn*:(*n*, Γ, (*Q*, *t*) # *cfg1*) $\in$ *cptn-mod-nest-call*
using *Catch.prems*(*1*) *Catch.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def* **by** *blast*
also then have ($\forall$ *m*<*n*. (*m*, Γ, (*Q*, *t*) # *cfg1*) $\notin$ *cptn-mod-nest-call*)
proof−
{ **fix** *m*
assume *ass*:*m*<*n*
{ **assume** *Q-m*:(*m*, Γ, (*Q*, *t*) # *cfg1*) $\in$ *cptn-mod-nest-call*
then have *t-eq-s*:*t*=*Normal s''* **using** *Catch catch2-ass* **by** *fastforce*

then obtain *xsa' s1 s1'* **where**
p0-cptn:(*m*, Γ,(*Q'*, *t*)#*xsa'*) $\in$ *cptn-mod-nest-call* **and**
catch-cond:*catch-cond-nest cfg1 P1 xsa' Q'* (*Normal s''*) *s1 s1'* Γ *m*

using *Q-m div-catch-nest*[*of m* Γ (*Q*, *t*) # *cfg1*] *Q′* **by** *blast*
have *fst*:*fst* (*last* ((*Q′*, *Normal s″*) # *xsa*)) = *LanguageCon.com.Throw*
using *catch2-ass Cons Q′* **by** (*simp add*: *last-length t-eq-s*)
have *cfg*:*cfg1* = *map* (*lift-catch P1*) *xsa* @ (*P1*, *snd* (*last* ((*Q′*, *Normal s″*) # *xsa*))) # *ys*
using *catch2-ass Cons Q′* **by** (*simp add*: *last-length t-eq-s*)
have *snd*:*snd* (*last* ((*Q′*, *Normal s″*) # *xsa*)) = *Normal s′*
using *catch2-ass Cons Q′* **by** (*simp add*: *last-length t-eq-s*)
then have *xsa*=*xsa′* ∧
(*m*, Γ, (*P1*, *snd* (((*Q′*, *Normal s″*) # *xsa*) ! *length xsa*)) # *ys*) ∈ *cptn-mod-nest-call*
using *catch2-ass Catch-P-Ends-Normal*[*OF cfg fst snd catch-cond*] *Cons*
by *auto*
then have *False* **using** *min-P1 ass Q′ t-eq-s* **unfolding** *min-call-def* **by** *auto*
}
} **then show** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
next
case *False*
then have *mp1*<*n* **using** *mp* **by** *auto*
then have *not-min-call-p1-n*:¬ *min-call n* Γ ((*P1*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*)
using *min-call-p1 last-length* **unfolding** *min-call-def* **by** *metis*
then have *min-call*:*min-call n* Γ ((*P0*, *s*) # *xs*)
using *min-call last-length* **unfolding** *min-call-def* **by** *metis*
then have (*P0*, *s*) # *xs* = (*P0*, *s*) # *xa*#*xsa*
using *Cons* **by** *auto*
then have *big*:*biggest-nest-call P0 s* (((*Q′*,*t*))#*xsa*) Γ *n*
proof−
have ¬(∃ *xs*. *min-call n* Γ ((*P0*, *s*)#*xs*) ∧ (*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs*)
using *min-call catch2-ass Cons*
proof −
have *min-call n* Γ ((*Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1*)
using *Catch.prems*(*1*) *Catch.prems*(*2*) **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *Catch-P-Not-finish append-Nil2 list.simps*(*3*)
same-append-eq catch catch2-ass)
qed
moreover have ¬(∃ *xs ys*. *cond-catch-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*)
unfolding *cond-catch-1-def* **using** *catch2-ass*
by (*metis Catch-P-Ends-Skip LanguageCon.com.distinct*(*17*) *catch last-length*)
moreover have ∃ *xs ys*. *cond-catch-2 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys s′ s″*
using *catch2-ass p0-cptn* **unfolding** *cond-catch-2-def last-length*

by *metis*

moreover have (*SOME xs*. $\exists$ *ys* s' s''. *cond-catch-2 n* Γ *P0 s xs P1* $((Q, t)$ # *cfg1*) *ys* s' s'') = *xs*

proof −

let *?P* = λ*xsa*. *s* = *Normal* s'' $\wedge$
(*n*, Γ, (*P0*, *s*) # *xsa*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xsa*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xsa*)) = *Normal* s' $\wedge$
(*n*, Γ, (*P1*, *Normal* s') # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xsa* @ (*P1*, *Normal* s') # *ys*

have ($\bigwedge$*x*. $\exists$ *ys* s' s''. *s* = *Normal* s'' $\wedge$
(*n*, Γ, (*P0*, *s*) # *x*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *x*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *x*)) = *Normal* s' $\wedge$
(*n*, Γ, (*P1*, *Normal* s') # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *x* @ (*P1*, *Normal* s') # *ys* $\Longrightarrow$
x = *xs*)

by (*metis Catch-P-Ends-Normal catch*)

moreover have $\exists$ *ys*. *s* = *Normal* s'' $\wedge$
(*n*, Γ, (*P0*, *s*) # *xs*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal* s' $\wedge$
(*n*, Γ, (*P1*, *Normal* s') # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* @ (*P1*, *Normal* s') # *ys*

using *ass p0-cptn* **by** (*metis* (*full-types*) *last-length*)

ultimately show *?thesis* **using** *some-equality*[*of ?P xs*]
unfolding *cond-catch-2-def* **by** *blast*

qed

moreover have (*SOME ys*. $\exists$ s' s''. *cond-catch-2 n* Γ *P0 s xs P1* $((Q, t)$ # *cfg1*) *ys* s' s'') = *ys*

proof −

let *?P* = λ*ysa*. *s* = *Normal* s'' $\wedge$
(*n*, Γ, (*P0*, *s*) # *xs*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal* s' $\wedge$
(*n*, Γ, (*P1*, *Normal* s') # *ysa*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* @ (*P1*, *Normal* s') # *ysa*

have ($\bigwedge$*x*. $\exists$ s' s''. *s* = *Normal* s'' $\wedge$
(*n*, Γ, (*P0*, *s*) # *xs*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal* s' $\wedge$
(*n*, Γ, (*P1*, *Normal* s') # *x*) $\in$ *cptn-mod-nest-call* $\wedge$ (*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* @ (*P1*, *Normal* s') # *x* $\Longrightarrow$
x = *ys*) **using** *catch2-ass* **by** *auto*

moreover have *s* = *Normal* s'' $\wedge$

$(n, \Gamma, (P0, s) \# xs) \in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Throw* $\wedge$
snd (*last* ((*P0*, *s*) # *xs*)) = *Normal* $s' \wedge$
$(n, \Gamma, (P1, Normal\ s') \# ys) \in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* @ (*P1*, *Normal* s') # *ys*
using *ass p0-cptn* **by** (*metis* (*full-types*) *catch2-ass last-length p0-cptn*)

ultimately show *?thesis* **using** *some-equality*[*of ?P ys*]
unfolding *cond-catch-2-def* **by** *blast*
qed
ultimately have *biggest-nest-call P0 s xs* Γ *n*
using *not-min-call-p1-n Catch*(*6*)
biggest-nest-call.simps(*2*)[*of P0 P1 s* (*Q*, *t*) # *cfg1* Γ *n*]
by *presburger*
then show *?thesis* **using** *Cons Q'* **by** *auto*
qed
have *C*:(*P0*, *s*) # *xs* = (*P0*, *s*) # (*Q'*, *t*) # *xsa* **using** *Cons Q'* **by** *auto*
have *reP0*:*redex P0* = (*Call f*) $\wedge$ Γ *f* = *Some bdy* $\wedge$
($\exists$ *saa*. *s* = *Normal saa*) $\wedge$ *t* = *s* **using** *Catch*(*5*) *Q'* **by** *auto*
then have *min-call*:*min-call n* Γ ((*Q'*, *t*) # *xsa*) **using** *Catch*(*1*)[*OF min-call C reP0 big*]
by *auto*
have *p1-n-cptn*:$(n, \Gamma, (Q, t) \# cfg1) \in$ *cptn-mod-nest-call*
using *Catch.prems*(*1*) *Catch.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def* **by** *blast*
also then have ($\forall m<n$. $(m, \Gamma, (Q, t) \# cfg1) \notin$ *cptn-mod-nest-call*)
proof−
{ fix *m*
assume *ass*:*m*<*n*
{ assume *Q-m*:$(m, \Gamma, (Q, t) \# cfg1) \in$ *cptn-mod-nest-call*
then have *t-eq-s*:*t*=*Normal* s'' **using** *Catch catch2-ass* **by** *fastforce*
then obtain *xsa' s1 s1'* **where**
p0-cptn:$(m, \Gamma, (Q', t) \# xsa') \in$ *cptn-mod-nest-call* **and**
catch-cond:*catch-cond-nest cfg1 P1 xsa' Q'* (*Normal* s'') *s1 s1'* Γ *m*
using *Q-m div-catch-nest*[*of m* Γ (*Q*, *t*) # *cfg1*] *Q'* **by** *blast*
have *fst*:*fst* (*last* ((*Q'*, *Normal* s'') # *xsa*)) = *LanguageCon.com.Throw*

using *catch2-ass Cons Q'* **by** (*simp add*: *last-length t-eq-s*)
have *cfg*:*cfg1* = *map* (*lift-catch P1*) *xsa* @ (*P1*, *snd* (*last* ((*Q'*, *Normal* s'') # *xsa*))) # *ys*
using *catch2-ass Cons Q'* **by** (*simp add*: *last-length t-eq-s*)
have *snd*:*snd* (*last* ((*Q'*, *Normal* s'') # *xsa*)) = *Normal* s'
using *catch2-ass Cons Q'* **by** (*simp add*: *last-length t-eq-s*)
then have *xsa*=*xsa'*
using *catch2-ass Catch-P-Ends-Normal*[*OF cfg fst snd catch-cond*] *Cons*
by *auto*
then have *False* **using** *min-call p0-cptn ass* **unfolding** *min-call-def* **by** *auto*

}
} **then show** *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed
qed
}**note** *l=this*
{**assume** *ass*:*fst* $(((P0, s) \# xs) ! length\ xs) = LanguageCon.com.Skip \wedge$
$(\exists ys. (n, \Gamma, (LanguageCon.com.Skip, snd (last ((P0, s) \# xs))) \# ys)$
$\in$ *cptn-mod-nest-call* $\wedge$
$(Q, t) \# cfg1 = map$ (*lift-catch P1*) xs @ $(LanguageCon.com.Skip, snd$
$(last ((P0, s) \# xs))) \# ys)$
have *?thesis*
proof (*cases* $\Gamma\vdash_c(Catch\ P0\ P1, s) \rightarrow (Q,t)$)
case *True*
thus *?thesis*
proof (*cases xs*)
case *Nil* **thus** *?thesis* **using** *Catch ass* **by** *fastforce*
next
case (*Cons xa xsa*)
then obtain *ys* **where**
catch2-ass:*fst* $(((P0, s) \# xs) ! length\ xs) = LanguageCon.com.Skip \wedge$
$(n, \Gamma, (LanguageCon.com.Skip, snd (last ((P0, s) \# xs))) \# ys) \in$
cptn-mod-nest-call $\wedge$
$(Q, t) \# cfg1 = map$ (*lift-catch P1*) xs @ $(LanguageCon.com.Skip, snd$
$(last ((P0, s) \# xs))) \# ys$
using *ass* **by** *auto*
then have *t-eq*:$t=s$ **using** *Catch* **by** *fastforce*
obtain *mq mp1* **where**
min-call-q:*min-call mq* Γ $((P0, s) \# xs)$ **and**
min-call-p1:*min-call mp1* Γ $((Skip, snd (((P0, s) \# xs) ! length\ xs)) \#$
$ys)$
using *catch2-ass minimum-nest-call p0-cptn* **by** (*metis last-length*)
then have *mp1-zero*:$mp1=0$ **by** (*simp add*: *skip-min-nested-call-0*)
then have *min-call*: *min-call n* Γ $((P0, s) \# xs)$
using *catch2-ass min-call-catch2*[*of n* Γ *P0 P1 s* $(Q, t) \# cfg1$ *xs ys*]
Catch(*3*,*4*) *p0-cptn* **by** (*metis last-length*)
have *n-z*:$n>0$ **using** *redex-call-cptn-mod-min-nest-call-gr-zero*[*OF Catch*(*3*)
Catch(*4*) *Catch*(*5*) *True*]
by *auto*
from *catch2-ass* **obtain** Q' **where** Q':$Q=Catch\ Q'\ P1 \wedge xa=(Q',t)$
unfolding *lift-catch-def* **using** *Cons*
proof −
assume *a1*: $\bigwedge Q'. Q = Catch\ Q'\ P1 \wedge xa = (Q', t) \Longrightarrow thesis$
have (*Catch* (*fst xa*) *P1*, *snd xa*) $= ((Q, t) \# cfg1) ! 0$
using *catch2-ass* **unfolding** *lift-catch-def*
by (*simp add*: *Cons case-prod-unfold*)
then show *?thesis*
using *a1* **by** *fastforce*
qed

have *big-call*:*biggest-nest-call P0 s* $((Q',t)\#xsa)$ Γ *n*
proof−
have $\neg(\exists$ *xs. min-call n* Γ $((P0, s)\#xs) \wedge (Q, t)$ # *cfg1* = *map* (*lift-catch P1*) *xs*)
using *min-call catch2-ass Cons*
proof −
have *min-call n* Γ ((*Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1*)
using *Catch.prems*(*1*) *Catch.prems*(*2*) **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *Catch-P-Not-finish append-Nil2 list.simps*(*3*) *same-append-eq catch catch2-ass*)
qed
moreover have ($\exists$ *xs ys. cond-catch-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*)
using *catch2-ass p0-cptn* **unfolding** *cond-catch-1-def last-length*
by *metis*
moreover have (*SOME xs.* $\exists$ *ys. cond-catch-1 n* Γ *P0 s xs P1* ((*Q*, *t*) # *cfg1*) *ys*) = *xs*
proof −
let *?P* = λ*xsa.* $\exists$ *ys.* (*n*, Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Skip* $\wedge$
(*n*, Γ, (*LanguageCon.com.Skip*,
snd (*last* ((*P0*, *s*) # *xsa*))) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xsa* @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xsa*))) # *ys*
have $\bigwedge$*xsa.* $\exists$ *ys.* (*n*, Γ,(*P0*, *s*)#*xsa*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Skip* $\wedge$
(*n*, Γ, (*LanguageCon.com.Skip*,
snd (*last* ((*P0*, *s*) # *xsa*))) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xsa* @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xsa*))) # *ys* $\Longrightarrow$
xsa = *xs*
using *Catch-P-Ends-Skip catch catch2-ass map-lift-catch-some-eq* **by** *fastforce*
moreover have $\exists$ *ys.* (*n*, Γ,(*P0*, *s*)#*xs*) $\in$ *cptn-mod-nest-call* $\wedge$
fst (*last* ((*P0*, *s*) # *xs*)) = *LanguageCon.com.Skip* $\wedge$
(*n*, Γ, (*LanguageCon.com.Skip*,
snd (*last* ((*P0*, *s*) # *xs*))) # *ys*) $\in$ *cptn-mod-nest-call* $\wedge$
(*Q*, *t*) # *cfg1* = *map* (*lift-catch P1*) *xs* @
(*LanguageCon.com.Skip*, *snd* (*last* ((*P0*, *s*) # *xs*))) # *ys*
using *ass p0-cptn* **by** (*simp add*: *last-length*)
ultimately show *?thesis* **using** *some-equality*[*of ?P xs*]
unfolding *cond-catch-1-def* **by** *blast*
qed
ultimately have *biggest-nest-call P0 s xs* Γ *n*
using *Catch*(*6*)
biggest-nest-call.simps(*2*)[*of P0 P1 s* (*Q*, *t*) # *cfg1* Γ *n*]
by *presburger*

then show *?thesis* **using** *Cons Q'* **by** *auto*
qed
have *min-call*:*min-call n* Γ ((*Q'*,*t*)#*xsa*)
using *Catch*(*1*)[*OF min-call - - big-call*] *Catch*(*5*) *Cons Q'* **by** *fastforce*

then have *p1-n-cptn*:(*n*, Γ, (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
using *Catch.prems*(*1*) *Catch.prems*(*2*) *elim-cptn-mod-nest-call-n min-call-def*
by *blast*
also then have (∀ *m*<*n*. (*m*, Γ, (*Q*, *t*) # *cfg1*) ∉ *cptn-mod-nest-call*)
proof−
{ **fix** *m*
assume *ass*:*m*<*n*
{ **assume** *Q-m*:(*m*, Γ, (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call*
then obtain *xsa' s1 s1'* **where**
p0-cptn:(*m*, Γ,(*Q'*, *t*)#*xsa'*) ∈ *cptn-mod-nest-call* **and**
seq:*catch-cond-nest cfg1 P1 xsa' Q' t s1 s1'* Γ *m*
using *div-catch-nest*[*of m* Γ (*Q*, *t*) # *cfg1*] *Q' t-eq* **by** *blast*
then have *xsa*=*xsa'*
using *catch2-ass*
Catch-P-Ends-Skip[*of cfg1 P1 xsa Q' t ys xsa' s1 s1'*]
Cons Q' Q-m
by (*simp add*: *last-length*)
then have *False* **using** *min-call p0-cptn ass* **unfolding** *min-call-def*
by *auto*
}
} **then show** *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **unfolding** *min-call-def* **by** *auto*
qed
next
case *False*
then have *env*:Γ$\vdash_c$(*Catch P0 P1*, *s*) $\rightarrow_e$ (*Q*,*t*) **using** *Catch*
by (*meson elim-cptn-mod-nest-step-c min-call-def*)
moreover then have *Q*:*Q*=*Catch P0 P1* **using** *env-c-c'* **by** *blast*
ultimately show *?thesis* **using** *Catch*
proof −
obtain *nn* :: (('*b*, '*a*, '*c*,'*d*) *LanguageCon.com* × ('*b*, '*c*) *xstate*) *list* ⇒ ('*a* ⇒ ('*b*, '*a*, '*c*,'*d*) *LanguageCon.com option*) ⇒ *nat* ⇒ *nat* **where**
f1: ∀ *x0 x1 x2*. (∃ *v3*<*x2*. (*v3*, *x1*, *x0*) ∈ *cptn-mod-nest-call*) = (*nn x0 x1 x2* < *x2* ∧ (*nn x0 x1 x2*, *x1*, *x0*) ∈ *cptn-mod-nest-call*)
by *moura*
have *f2*: (*n*, Γ, (*LanguageCon.com.Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1*) ∈ *cptn-mod-nest-call* ∧ (∀ *n*. ¬ *n* < *n* ∨ (*n*, Γ, (*LanguageCon.com.Catch P0 P1*, *s*) # (*Q*, *t*) # *cfg1*) ∉ *cptn-mod-nest-call*)
using *local.Catch*(*3*) *local.Catch*(*4*) *min-call-def* **by** *blast*
then have ¬ *nn* ((*Q*, *t*) # *cfg1*) Γ *n* < *n* ∨ (*nn* ((*Q*, *t*) # *cfg1*) Γ *n*, Γ, (*Q*, *t*) # *cfg1*) ∉ *cptn-mod-nest-call*
using *False env env-c-c' not-func-redex-cptn-mod-nest-n-env*
by (*metis Catch.prems*(*1*) *Catch.prems*(*2*) *min-call-def*)
then show *?thesis*

using *f2 f1* **by** (*meson elim-cptn-mod-nest-call-n min-call-def*)
qed
qed
}
thus *?thesis* **using** *l ass* **by** *fastforce*
qed
qed (*fastforce*)+

lemma *cptn-mod-nest-n-1*:
assumes *a0*:$(n,\Gamma,cfs) \in$ *cptn-mod-nest-call* **and**
a1:$cfs=(p,s)\#cfs'$ **and**
a2:$\neg$ (*min-call n* Γ *cfs*)
shows $(n-1,\Gamma,cfs) \in$ *cptn-mod-nest-call*
using *a0 a1 a2*
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leI cptn-mod-nest-mono less-nat-zero-code min-call-def not-less*)

lemma *cptn-mod-nest-tl-n-1*:
assumes *a0*:$(n,\Gamma,cfs) \in$ *cptn-mod-nest-call* **and**
a1:$cfs=(p,s)\#(q,t)\#cfs'$ **and**
a2:$\neg$ (*min-call n* Γ *cfs*)
shows $(n-1,\Gamma,(q,t)\#cfs') \in$ *cptn-mod-nest-call*
using *a0 a1 a2*
by (*meson elim-cptn-mod-nest-call-n cptn-mod-nest-n-1*)

lemma *cptn-mod-nest-tl-not-min*:
assumes *a0*:$(n,\Gamma,cfg) \in$ *cptn-mod-nest-call* **and**
a1:$cfg=(p,s)\#cfg'$ **and**
a2:$\neg$ (*min-call n* Γ *cfg*)
shows $\neg$ (*min-call n* Γ *cfg'*)
proof (*cases cfg'*)
case *Nil*
have $(\Gamma, []) \notin$ *cptn*
using *cptn.simps* **by** *auto*
then show *?thesis* **unfolding** *min-call-def*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod local.Nil* **by** *blast*
next
case (*Cons xa cfga*)
then obtain *q t* **where** $xa = (q,t)$ **by** *fastforce*
then have $(n-1,\Gamma,cfg') \in$ *cptn-mod-nest-call*
using *a0 a1 a2 cptn-mod-nest-tl-n-1 Cons* **by** *fastforce*
also then have $(n,\Gamma,cfg') \in$ *cptn-mod-nest-call*
using *cptn-mod-nest-mono Nat.diff-le-self* **by** *blast*
ultimately show *?thesis* **unfolding** *min-call-def*
using *a0 a2 min-call-def* **by** *force*
qed

definition *cpn* :: *nat* ⇒ ('*s*,'*p*,'*f*,'*e*) *body* ⇒ ('*s*,'*p*,'*f*,'*e*) *com* ⇒
('*s*,'*f*) *xstate* ⇒ (('*s*,'*p*,'*f*,'*e*) *confs*) *set*
where
cpn n Γ *P s* ≡ {(Γ*1*,*l*). *l*!*0*=(*P*,*s*) ∧ (*n*,Γ,*l*) ∈ *cptn-mod-nest-call* ∧ Γ*1*=Γ}

lemma *cptn-mod-same-n*:
assumes *a0*:(Γ,*cfs*)∈ *cptn-mod*
shows ∃ *n*. (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call*
proof −
show *?thesis* **using** *cptn-mod-nest-mono cptn-mod-cptn-mod-nest*
by (*metis a0 cptn-mod-nest-mono2 leI*)
qed

lemma *cptn-mod-same-n1*:
assumes *a0*:(Γ,*cfs*)∈ *cptn-mod* **and**
a1:(Γ,*cfs1*)∈ *cptn-mod*
shows ∃ *n*. (*n*,Γ,*cfs*) ∈ *cptn-mod-nest-call* ∧ (*n*,Γ,*cfs1*) ∈ *cptn-mod-nest-call*
proof −
show *?thesis* **using** *cptn-mod-nest-mono cptn-mod-cptn-mod-nest*
by (*metis a0 a1 cptn-mod-nest-mono2 leI*)
qed

lemma *dropcptn-is-cptn1* [*rule-format*,*elim*!]:
∀ *j*<*length c*. (*n*,Γ,*c*) ∈ *cptn-mod-nest-call* ⟶ (*n*,Γ, *drop j c*) ∈ *cptn-mod-nest-call*
proof −
{**fix** *j*
assume *j*<*length c* ∧ (*n*,Γ,*c*) ∈ *cptn-mod-nest-call*
then have (*n*,Γ, *drop j c*) ∈ *cptn-mod-nest-call*
proof(*induction j arbitrary*: *c*)
case *0* **then show** *?case* **by** *auto*
next
case (*Suc j*)
then obtain *a b c'* **where** *c*=*a*#*b*#*c'*
by (*metis Cons-nth-drop-Suc Suc-lessE drop-0 less-trans-Suc zero-less-Suc*)
then also have *j*<*length* (*b*#*c'*) **using** *Suc* **by** *auto*
ultimately moreover have (*n*, Γ, *drop j* (*b* # *c'*)) ∈ *cptn-mod-nest-call*
using *elim-cptn-mod-nest-call-n*[*of n* Γ *c*] *Suc*
by (*metis surj-pair*)
ultimately show *?case* **by** *auto*
qed
} **thus** *?thesis* **by** *auto*
qed

8.13 Compositionality of the Semantics

8.13.1 Definition of the conjoin operator

definition $\mathit{same\text{-}length} :: ('s,'p,'f,'e)\ \mathit{par\text{-}confs} \Rightarrow (('s,'p,'f,'e)\ \mathit{confs})\ \mathit{list} \Rightarrow \mathit{bool}$
where
$\mathit{same\text{-}length}\ c\ \mathit{clist} \equiv (\forall\, i<\mathit{length}\ \mathit{clist}.\ \mathit{length}(\mathit{snd}\ (\mathit{clist}!i))=\mathit{length}\ (\mathit{snd}\ c))$

lemma *same-length-non-pair*:
assumes $a1$:$\mathit{same\text{-}length}\ c\ \mathit{clist}$ **and**
$a2$:$\mathit{clist}'=\mathit{map}\ (\lambda x.\ \mathit{snd}\ x)\ \mathit{clist}$
shows $(\forall\, i <\mathit{length}\ \mathit{clist}'.\ \mathit{length}(\ (\mathit{clist}'!i))=\mathit{length}\ (\mathit{snd}\ c))$
using *a1 a2* **by** (*auto simp add*: *same-length-def*)

definition $\mathit{same\text{-}state} :: ('s,'p,'f,'e)\ \mathit{par\text{-}confs} \Rightarrow (('s,'p,'f,'e)\ \mathit{confs})\ \mathit{list} \Rightarrow \mathit{bool}$
where
$\mathit{same\text{-}state}\ c\ \mathit{clist} \equiv (\forall\, i <\mathit{length}\ \mathit{clist}.\ \forall\, j<\mathit{length}\ (\mathit{snd}\ c).\ \mathit{snd}((\mathit{snd}\ c)!j) = \mathit{snd}((\mathit{snd}\ (\mathit{clist}!i))!j))$

lemma *same-state-non-pair*:
assumes $a1$:$\mathit{same\text{-}state}\ c\ \mathit{clist}$ **and**
$a2$:$\mathit{clist}'=\mathit{map}\ (\lambda x.\ \mathit{snd}\ x)\ \mathit{clist}$
shows $(\forall\, i <\mathit{length}\ \mathit{clist}'.\ \forall\, j<\mathit{length}\ (\mathit{snd}\ c).\ \mathit{snd}((\mathit{snd}\ c)!j) = \mathit{snd}(\ (\mathit{clist}'!i)!j))$
using *a1 a2* **by** (*auto simp add*: *same-state-def*)

definition $\mathit{same\text{-}program} :: ('s,'p,'f,'e)\ \mathit{par\text{-}confs} \Rightarrow (('s,'p,'f,'e)\ \mathit{confs})\ \mathit{list} \Rightarrow \mathit{bool}$
where
$\mathit{same\text{-}program}\ c\ \mathit{clist} \equiv (\forall\, j<\mathit{length}\ (\mathit{snd}\ c).\ \mathit{fst}((\mathit{snd}\ c)!j) = \mathit{map}\ (\lambda x.\ \mathit{fst}(\mathit{nth}\ (\mathit{snd}\ x)\ j))\ \mathit{clist})$

lemma *same-program-non-pair*:
assumes $a1$:$\mathit{same\text{-}program}\ c\ \mathit{clist}$ **and**
$a2$:$\mathit{clist}'=\mathit{map}\ (\lambda x.\ \mathit{snd}\ x)\ \mathit{clist}$
shows $(\forall\, j<\mathit{length}\ (\mathit{snd}\ c).\ \mathit{fst}((\mathit{snd}\ c)!j) = \mathit{map}\ (\lambda x.\ \mathit{fst}(\mathit{nth}\ x\ j))\ \mathit{clist}')$
using *a1 a2* **by** (*auto simp add*: *same-program-def*)

definition $\mathit{same\text{-}functions} :: ('s,'p,'f,'e)\ \mathit{par\text{-}confs} \Rightarrow (('s,'p,'f,'e)\ \mathit{confs})\ \mathit{list} \Rightarrow \mathit{bool}$ **where**
$\mathit{same\text{-}functions}\ c\ \mathit{clist} \equiv \forall\, i <\mathit{length}\ \mathit{clist}.\ \mathit{fst}\ (\mathit{clist}!i) = \mathit{fst}\ c$

definition $\mathit{compat\text{-}label} :: ('s,'p,'f,'e)\ \mathit{par\text{-}confs} \Rightarrow (('s,'p,'f,'e)\ \mathit{confs})\ \mathit{list} \Rightarrow \mathit{bool}$
where
$\mathit{compat\text{-}label}\ c\ \mathit{clist} \equiv$
$(\forall\, j.\ \mathit{Suc}\ j<\mathit{length}\ (\mathit{snd}\ c) \longrightarrow$
$(\ ((\mathit{fst}\ c)\vdash_p((\mathit{snd}\ c)!j)\ \rightarrow ((\mathit{snd}\ c)!(\mathit{Suc}\ j))) \wedge$
$(\exists\, i<\mathit{length}\ \mathit{clist}.$
$((\mathit{fst}\ (\mathit{clist}!i))\vdash_c ((\mathit{snd}\ (\mathit{clist}!i))!j)\ \rightarrow ((\mathit{snd}\ (\mathit{clist}!i))!(\mathit{Suc}\ j))) \wedge$
$(\forall\, l<\mathit{length}\ \mathit{clist}.$
$l\neq i \longrightarrow\ (\mathit{fst}\ (\mathit{clist}!l))\vdash_c (\mathit{snd}\ (\mathit{clist}!l))!j\ \rightarrow_e ((\mathit{snd}\ (\mathit{clist}!l))!(\mathit{Suc}\ j))$

))) ∨
        $((fst\ c)\vdash_p((snd\ c)!j)\ \ \rightarrow_e\ ((snd\ c)!(Suc\ j))\ \wedge$
        $(\forall\, i{<}length\ clist.\ \ (fst\ (clist!i))\vdash_c\ (snd\ (clist!i))!j\ \ \rightarrow_e\ ((snd\ (clist!i))!(Suc\ j))\ \ )))$

lemma *compat-label-tran-0*:
 assumes *assm1*:$compat\text{-}label\ c\ clist \wedge length\ (snd\ c) > Suc\ 0$
 shows $((fst\ c)\vdash_p((snd\ c)!0)\ \ \rightarrow ((snd\ c)!(Suc\ 0))) \vee$
   $((fst\ c)\vdash_p((snd\ c)!0)\ \ \rightarrow_e\ ((snd\ c)!(Suc\ 0)))$
  using *assm1* **unfolding** *compat-label-def*
 by *blast*

definition *conjoin* :: $(('s,'p,'f,'e)\ par\text{-}confs) \Rightarrow (('s,'p,'f,'e)\ confs)\ list \Rightarrow bool$ (- ∝ - [65,65] 64) **where**
 $c \propto clist \equiv (same\text{-}length\ c\ clist) \wedge (same\text{-}state\ c\ clist) \wedge (same\text{-}program\ c\ clist) \wedge$
        $(compat\text{-}label\ c\ clist) \wedge (same\text{-}functions\ c\ clist)$

lemma *conjoin-same-length*:
  $c \propto clist \Longrightarrow \forall\, i < length\ (snd\ c).\ length\ (fst\ ((snd\ c)!i)) = \ length\ clist$
proof (*auto*)
 fix *i*
 assume *a1*:$c \propto clist$
 assume *a2*:$i < length\ (snd\ c)$
 then have $(\forall\, j{<}length\ (snd\ c).\ fst((snd\ c)!j) = map\ (\lambda x.\ fst(nth\ (snd\ x)\ j))\ clist)$
  using *a1* **unfolding** *conjoin-def same-program-def* **by** *auto*
 thus $length\ (fst\ (snd\ c\ !\ i)) = length\ clist$ **by** (*simp add*: *a2*)
qed

lemma $c \propto clist \Longrightarrow$
    $i{<}\ length\ (snd\ c) \wedge j < length\ (snd\ c) \Longrightarrow$
    $length\ (fst\ ((snd\ c)!i)) = length\ (fst\ ((snd\ c)!j))$
using *conjoin-same-length* **by** *fastforce*

lemma *conjoin-same-length-i-suci*:$c \propto clist \Longrightarrow$
    $Suc\ i{<}\ length\ (snd\ c) \Longrightarrow$
    $length\ (fst\ ((snd\ c)!i)) = length\ (fst\ ((snd\ c)!(Suc\ i)))$
using *conjoin-same-length* **by** *fastforce*

lemma *conjoin-same-program-i*:
 $c \propto clist \Longrightarrow$
  $j < length\ (snd\ c) \Longrightarrow$
  $i < length\ clist \Longrightarrow$
  $fst\ ((snd\ (clist!i))!j) = (fst\ ((snd\ c)!j))!i$
proof −
 assume *a0*:$c \propto clist$ **and**

a1:*j* < *length* (*snd* *c*) **and**
a2:*i* < *length* *clist*
have *length* (*fst* ((*snd* *c*)!*j*)) = *length* *clist*
using *conjoin-same-length* *a0* *a1* **by** *fastforce*
also have *fst* (*snd* *c* ! *j*) = *map* (λ*x*. *fst* (*snd* *x* ! *j*)) *clist*
using *a0* *a1* **unfolding** *conjoin-def* *same-program-def* **by** *fastforce*
ultimately show *?thesis* **using** *a2* **by** *fastforce*
qed

lemma *conjoin-same-program-i-j*:
c ∝ *clist* ⟹
Suc *j* < *length* (*snd* *c*) ⟹
∀ *l*< *length* *clist*. *fst* ((*snd* (*clist*!*l*))!*j*) = *fst* ((*snd* (*clist*!*l*))!(*Suc* *j*)) ⟹
fst ((*snd* *c*)!*j*) = (*fst* ((*snd* *c*)!(*Suc* *j*)))
proof −
assume *a0*:*c* ∝ *clist* **and**
a1:*Suc* *j* < *length* (*snd* *c*) **and**
a2:∀ *l*< *length* *clist*. *fst* ((*snd* (*clist*!*l*))!*j*) = *fst* ((*snd* (*clist*!*l*))!(*Suc* *j*))
have *length* (*fst* ((*snd* *c*)!*j*)) = *length* *clist*
using *conjoin-same-length* *a0* *a1* **by** *fastforce*
then have *map* (λ*x*. *fst* (*snd* *x* ! *j*)) *clist* = *map* (λ*x*. *fst* (*snd* *x* ! (*Suc* *j*))) *clist*
using *a2* **by** (*metis* (*no-types*, *lifting*) *in-set-conv-nth* *map-eq-conv*)
moreover have *fst* (*snd* *c* ! *j*) = *map* (λ*x*. *fst* (*snd* *x* ! *j*)) *clist*
using *a0* *a1* **unfolding** *conjoin-def* *same-program-def* **by** *fastforce*
moreover have *fst* (*snd* *c* ! *Suc* *j*) = *map* (λ*x*. *fst* (*snd* *x* ! *Suc* *j*)) *clist*
using *a0* *a1* **unfolding** *conjoin-def* *same-program-def* **by** *fastforce*
ultimately show *?thesis* **by** *fastforce*
qed

lemma *conjoin-last-same-state*:
assumes *a0*: (Γ,*l*)∝ *clist* **and**
a1: *i* < *length* *clist* **and**
a2: (*snd* (*clist*!*i*))≠[]
shows *snd* (*last* (*snd* (*clist*!*i*))) = *snd* (*last* *l*)
proof −
have *length* *l* = *length* (*snd* (*clist*!*i*))
using *a0* *a1* **unfolding** *conjoin-def* *same-length-def* **by** *fastforce*
also then have *length-l*:*length* *l* ≠*0* **using** *a2* **by** *fastforce*
ultimately have *last* (*snd* (*clist*!*i*)) = (*snd* (*clist*!*i*))!((*length* *l*)−*1*)
using *a1* *a2*
by (*simp* *add*: *last-conv-nth*)
thus *?thesis* **using** *length-l* *a0* *a1* **unfolding** *conjoin-def* *same-state-def*
by (*simp* *add*: *a2* *last-conv-nth*)
qed

lemma *list-eq-if* [*rule-format*]:
∀ *ys*. *xs*=*ys* ⟶ (*length* *xs* = *length* *ys*) ⟶ (∀ *i*<*length* *xs*. *xs*!*i*=*ys*!*i*)
by (*induct* *xs*) *auto*

lemma *list-eq*: $(length\ xs = length\ ys \wedge (\forall i{<}length\ xs.\ xs!i{=}ys!i)) = (xs{=}ys)$
apply(*rule iffI*)
 apply *clarify*
 apply(*erule nth-equalityI*)
 apply *simp+*
done

lemma *nth-tl*: $\llbracket\ ys!0{=}a;\ ys{\neq}[]\ \rrbracket \Longrightarrow ys{=}(a\#(tl\ ys))$
  by (*cases ys*) *simp-all*

lemma *nth-tl-if* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ ys \longrightarrow P\ (a\#(tl\ ys))$
  by (*induct ys*) *simp-all*

lemma *nth-tl-onlyif* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ (a\#(tl\ ys)) \longrightarrow P\ ys$
  by (*induct ys*) *simp-all*

lemma *nth-tl-eq* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ (a\#(tl\ ys)) = P\ ys$
  by (*induct ys*) *simp-all*

lemma *nth-tl-pair*: $\llbracket p{=}(u,ys);\ ys!0{=}a;\ ys{\neq}[]\ \rrbracket \Longrightarrow p{=}(u,(a\#(tl\ ys)))$
by (*simp add*: *SmallStepCon.nth-tl*)

lemma *nth-tl-eq-Pair* [*rule-format*]: $p{=}(u,ys) \longrightarrow ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ ((u,a\#(tl\ ys))) = P\ (u,ys)$
  by (*induct ys*) *simp-all*

lemma *tl-in-cptn*: $\llbracket\ (g,a\#xs) \in cptn;\ xs{\neq}[]\ \rrbracket \Longrightarrow (g,xs){\in}cptn$
  by (*force elim*: *cptn.cases*)

lemma *tl-zero*[*rule-format*]:
   $Suc\ j{<}length\ ys \longrightarrow P\ (ys!Suc\ j) \longrightarrow P\ (tl(ys)!j)$
  by (*simp add*: *List.nth-tl*)

lemma *tl-zero1*[*rule-format*]:
  $Suc\ j{<}length\ ys \longrightarrow P\ (tl(ys)!j) \longrightarrow P\ (ys!Suc\ j)$
 by (*simp add*: *List.nth-tl*)

lemma *tl-zero-eq* [*rule-format*]:
  $Suc\ j{<}length\ ys \longrightarrow (P\ (tl(ys)!j) = P\ (ys!Suc\ j))$
by (*simp add*: *List.nth-tl*)

lemma *tl-zero-eq'* :
   $\forall j.\ Suc\ j{<}length\ ys \longrightarrow (P\ (tl(ys)!j) = P\ (ys!Suc\ j))$
using *tl-zero-eq* **by** *blast*

lemma $tl\text{-}zero\text{-}pair$:$i < length\ ys \Longrightarrow length\ ys = length\ zs \Longrightarrow$
$Suc\ j < length\ (snd\ (ys!i)) \Longrightarrow$
$snd\ (zs!i) = tl\ (snd\ (ys!i)) \Longrightarrow$
$P\ ((snd\ (ys!i))!(Suc\ j)) =$
$P\ ((snd\ (zs!i))!j)$
by ($simp\ add$: $tl\text{-}zero\text{-}eq$)

lemma $tl\text{-}zero\text{-}pair'$:$\forall\, i < length\ ys.\ length\ ys = length\ zs \longrightarrow$
$Suc\ j < length\ (snd\ (ys!i)) \longrightarrow$
$snd\ (zs!i) = tl\ (snd\ (ys!i)) \longrightarrow$
$(P\ ((snd\ (ys!i))!(Suc\ j)) =$
$P\ ((snd\ (zs!i))!j))$
using $tl\text{-}zero\text{-}pair$ **by** $blast$

lemma $tl\text{-}zero\text{-}pair2$:$i < length\ ys \Longrightarrow length\ ys = length\ zs \Longrightarrow$
$Suc\ (Suc\ j) < length\ (snd\ (ys!i)) \Longrightarrow$
$snd\ (zs!i) = tl\ (snd\ (ys!i)) \Longrightarrow$
$P\ ((snd\ (ys!i))!(Suc\ (Suc\ j)))\ ((snd\ (ys!i))!(Suc\ j)) =$
$P\ ((snd\ (zs!i))!(Suc\ j))\ ((snd\ (zs!i))!j)$
by ($simp\ add$: $tl\text{-}zero\text{-}eq$)

lemma $tl\text{-}zero\text{-}pair2'$:$\forall\, i < length\ ys.\ length\ ys = length\ zs \longrightarrow$
$Suc\ (Suc\ j) < length\ (snd\ (ys!i)) \longrightarrow$
$snd\ (zs!i) = tl\ (snd\ (ys!i)) \longrightarrow$
$P\ ((snd\ (ys!i))!(Suc\ (Suc\ j)))\ ((snd\ (ys!i))!(Suc\ j)) =$
$P\ ((snd\ (zs!i))!(Suc\ j))\ ((snd\ (zs!i))!j)$
using $tl\text{-}zero\text{-}pair2$ **by** $blast$

lemma $tl\text{-}zero\text{-}pair21$:$\forall\, i < length\ ys.\ length\ ys = length\ zs \longrightarrow$
$Suc\ (Suc\ j) < length\ (snd\ (ys!i)) \longrightarrow$
$snd\ (zs!i) = tl\ (snd\ (ys!i)) \longrightarrow$
$P\ ((snd\ (ys!i))!(Suc\ j))\ ((snd\ (ys!i))!(Suc\ (Suc\ j)))=$
$P\ ((snd\ (zs!i))!j)\ ((snd\ (zs!i))!(Suc\ j))$
by ($metis\ SmallStepCon.nth\text{-}tl\ list.size(3)\ not\text{-}less0\ nth\text{-}Cons\text{-}Suc$)

lemma $tl\text{-}pair$:$Suc\ (Suc\ j) < length\ l \Longrightarrow$
$l1 = tl\ l \Longrightarrow$
$P\ (l!(Suc\ (Suc\ j)))\ (l!(Suc\ j)) =$
$P\ (l1!(Suc\ j))\ (l1!j)$
by ($simp\ add$: $tl\text{-}zero\text{-}eq$)

lemma $list\text{-}as\text{-}map$:
assumes
$a1$:$length\ clist > 0$ **and**
$a2$: $xs = (map\ (\lambda x.\ fst\ (hd\ x))\ clist)$ **and**
$a3$: $ys = (map\ (\lambda x.\ tl\ x)\ clist)$ **and**
$a4$: $\forall\, i< length\ clist.\ length\ (clist!i) > 0$ **and**

```
    a5: ∀ i < length clist. ∀ j< length clist. ∀ k<length (clist!i).
          snd ((clist!i)!k) = snd ((clist!j)!k) and
    a6: ∀ i < length clist. ∀ j< length clist.
          length (clist!i) = length (clist!j)
    shows clist = map (λi. (fst i,snd ((clist!0)!0))#snd i) (zip xs ys)
proof−
  let ?clist'= map (λi. (fst i,snd ((clist!0)!0))#snd i) (zip xs ys)
  have lens:length clist = length ?clist' using a2 a3 by auto
  have (∀ i<length clist. clist ! i = ?clist' ! i)
  proof −
    {
      fix i
      assume a11:i<length clist
      have xs-clist:xs!i = fst (hd (clist!i)) using a2 a11 by auto
      have ys-clist:ys!i = tl (clist ! i) using a3 a11 by auto
      have snd-zero:snd (hd (clist!i)) = snd ((clist!0)!0) using a5 a4
           by (metis (no-types, lifting) a1 a11 hd-conv-nth less-numeral-extra(3)
list.size(3))
        then have (λi. (fst i,snd ((clist!0)!0))#snd i) ((zip xs ys)!i) = clist !i

        proof −
          have f1: length xs = length clist
            using a2 length-map by blast
          have ¬ (0::nat) < 0
            by (meson less-not-refl)
          thus ?thesis
            using f1 by (metis (lifting) a11 a3 a4
                         fst-conv length-map list.exhaust-sel
                         list.size(3) nth-zip prod.collapse
                         snd-conv snd-zero xs-clist ys-clist)
        qed
      then have clist ! i = ?clist' ! i using lens a11 by force
    }
    thus ?thesis by auto
  qed
  thus ?thesis using lens list-eq by blast
qed

lemma list-as-map':
  assumes
    a1:length clist > 0 and
    a2: xs = (map (λx. hd x) clist) and
    a3: ys = (map (λx. tl x) clist) and
    a4: ∀ i< length clist. length (clist!i) > 0
    shows clist = map (λi. (fst i)#snd i) (zip xs ys)
proof−
  let ?clist'= map (λi.(fst i)#snd i) (zip xs ys)
  have lens:length clist = length ?clist' using a2 a3 by auto
  have (∀ i<length clist. clist ! i = ?clist' ! i)
```

```
  proof -
    {
      fix i
      assume a11:i<length clist
      have xs-clist:xs!i = hd (clist!i) using a2 a11 by auto
      have ys-clist:ys!i = tl (clist ! i) using a3 a11 by auto
      then have (λi. fst i#snd i) ((zip xs ys)!i) = clist !i
        using xs-clist ys-clist a11 a2 a3 a4 by fastforce
      then have clist ! i = ?clist' ! i using lens a11 by force
    }
    thus ?thesis by auto
  qed
  thus ?thesis using lens list-eq by blast
qed


lemma conjoin-tl:
  assumes
    a1: (Γ,x#xs) ∝ ys and
    a2:zs = map (λi. (fst i, tl (snd i))) ys
   shows (Γ,xs) ∝ zs
proof -
  have s-p:same-program (Γ,x#xs) ys using a1 unfolding conjoin-def by simp
  have s-l:same-length (Γ,x#xs) ys using a1 unfolding conjoin-def by simp
  have ∀i<length zs. snd (zs!i) = tl (snd (ys!i))
    by (simp add: a2)
  {
    have same-length (Γ,xs) zs using a1 a2 unfolding conjoin-def
     by (simp add: same-length-def)
  } moreover note same-len = this
  {
    {
       fix j
      assume a11:j<length (snd (Γ, xs))
       then have fst-suc:fst (snd (Γ, xs) ! j) = fst(snd (Γ,x#xs)! Suc j)
         by auto
       then have fst (snd (Γ, xs) ! j) = map (λx. fst (snd x ! j)) zs
       proof -
         have s-l-y-z:length ys = length zs using a2 by fastforce
         have Suc-j-l-ys:∀i < length ys. Suc j < length (snd (ys!i))
           using a11 s-l unfolding same-length-def by fastforce
         have tail: ∀i < length ys. snd (zs!i) = tl (snd (ys!i)) using a2
           by fastforce
         then have l-xs-zs-eq:length (fst (snd (Γ, xs) ! j)) = length zs
          using fst-suc s-l-y-z s-p a11 unfolding same-program-def by auto
         then have ∀i<length ys.
          fst (snd (Γ, x#xs) ! Suc j)!i = fst (snd (ys!i) ! (Suc j))
            using s-p a11 unfolding same-program-def by fastforce
         then have ∀i<length zs.
```

```
        fst (snd (Γ, x#xs) ! Suc j)!i = fst (snd (zs!i) ! (j))
        using Suc-j-l-ys tail s-l-y-z tl-zero-pair by metis
      then have ∀ i<length zs.
        fst (snd (Γ, xs) ! j)!i = map (λx. fst (snd x ! j)) zs!i
       using fst-suc by auto
      also have length (fst (snd (Γ, xs) ! j)) =
               length (map (λx. fst (snd x ! j)) zs)
       using l-xs-zs-eq by auto
      ultimately show ?thesis using l-xs-zs-eq list-eq by metis
     qed
  }
  then have same-program (Γ,xs) zs
  unfolding conjoin-def same-program-def same-length-def
  by blast
}moreover note same-prog = this
{
  have same-state (Γ,xs) zs
  using a1 a2 unfolding conjoin-def same-length-def same-state-def
  apply auto
 by (metis (no-types, hide-lams) List.nth-tl Suc-less-eq diff-Suc-1 length-tl nth-Cons-Suc)

}moreover note same-sta = this
{
  have same-functions (Γ,xs) zs
   using a1 a2 unfolding conjoin-def
   apply auto
   apply (simp add: same-functions-def)
   done
}moreover note same-fun = this
{ {
    fix j
    assume a11:Suc j<length (snd (Γ, xs))
    have s-l-y-z:length ys = length zs using a2 by fastforce
    have Suc-j-l-ys:∀ i < length ys. Suc (Suc j) < length (snd (ys!i))
      using a11 s-l unfolding same-length-def by fastforce
    have tail: ∀ i < length ys. snd (zs!i) = tl (snd (ys!i)) using a2
      by fastforce
    have same-env: ∀ i < length ys. (fst (ys!i)) = Γ
      using a1 unfolding conjoin-def same-functions-def by auto
    have fst: ∀ x. fst(Γ, x) = Γ by auto
    then have fun-ys-eq-fun-zs: ∀ i < length ys. (fst (ys!i)) = (fst (zs!i))
      using same-env s-l-y-z
      proof −
        have ∀ n. ¬ n < length ys ∨ fst (zs ! n) = fst (ys ! n)
          by (simp add: a2)
        thus ?thesis
          by presburger
      qed
    have suc-j:Suc (Suc j) < length (snd (Γ, x#xs)) using a11 by auto
```

then have *or-compat*:($(\Gamma \vdash_p((snd\ \ (\Gamma,\ x\#xs))!(Suc\ j))\ \ \rightarrow ((snd\ \ (\Gamma,\ x\#xs))!(Suc\ (Suc\ j)))) \wedge$
$(\exists\, i<length\ ys.$
$((fst\ (ys!i))\vdash_c ((snd\ (ys!i))!(Suc\ j))\ \rightarrow ((snd\ (ys!i))!(Suc\ (Suc\ j))))\ \wedge$
$(\forall\, l<length\ ys.$
$l\neq i \longrightarrow (fst\ (ys!l))\vdash_c (snd\ (ys!l))!(Suc\ j)\ \rightarrow_e ((snd\ (ys!l))!(Suc\ (Suc\ j)))\ \ )))\ \vee$
$(\Gamma\vdash_p((snd\ \ (\Gamma,\ x\#xs))!(Suc\ j))\ \rightarrow_e ((snd\ \ (\Gamma,\ x\#xs))!(Suc\ (Suc\ j)))\ \wedge$
$(\forall\, i<length\ ys.\ (fst\ (ys!i))\vdash_c (snd\ (ys!i))!(Suc\ j)\ \rightarrow_e ((snd\ (ys!i))!(Suc\ (Suc\ j)))))$
using *suc-j a1 same-env* **unfolding** *conjoin-def compat-label-def fst* **by** *auto*
then have
$(\ (fst\ (\Gamma,\ xs)\ \vdash_p((snd\ \ (\Gamma,\ xs))!(j))\ \ \rightarrow ((snd\ \ (\Gamma,xs))!((Suc\ j))))\ \wedge$
$(\exists\, i<length\ zs.$
$((fst\ (zs!i))\vdash_c ((snd\ (zs!i))!(\ j))\ \ \rightarrow ((snd\ (zs!i))!(\ (Suc\ j))))\ \wedge$
$(\forall\, l<length\ zs.$
$l\neq i \longrightarrow (fst\ (zs!l))\vdash_c (snd\ (zs!l))!(\ j)\ \ \rightarrow_e ((\ snd\ (zs!l))!(\ (Suc\ j)))\ )))\vee$
$((fst\ (\Gamma,\ xs)\vdash_p((snd\ \ (\Gamma,\ xs))!(j))\ \ \rightarrow_e ((snd\ \ (\Gamma,\ xs))!((Suc\ j)))\ \wedge$
$(\forall\, i<length\ zs.\ (fst\ (zs!i))\vdash_c (snd\ (zs!i))!(j)\ \ \rightarrow_e ((snd\ (zs!i))!((Suc\ j)))\ )))$
proof
assume *a21*:($(\Gamma \vdash_p((snd\ \ (\Gamma,\ x\#xs))!(Suc\ j))\ \ \rightarrow ((snd\ \ (\Gamma,\ x\#xs))!(Suc\ (Suc\ j))))\ \wedge$
$(\exists\, i<length\ ys.$
$((fst\ (ys!i))\vdash_c ((snd\ (ys!i))!(Suc\ j))\ \ \rightarrow ((snd\ (ys!i))!(Suc\ (Suc\ j))))\ \wedge$
$(\forall\, l<length\ ys.$
$l\neq i \longrightarrow (fst\ (ys!l))\vdash_c (snd\ (ys!l))!(Suc\ j)\ \ \rightarrow_e ((snd\ (ys!l))!(Suc\ (Suc\ j)))\ \ )))$
then obtain i **where**
f1:($(\Gamma \vdash_p((snd\ \ (\Gamma,\ x\#xs))!(Suc\ j))\ \ \rightarrow ((snd\ \ (\Gamma,\ x\#xs))!(Suc\ (Suc\ j))))\ \wedge$
$(i<length\ ys\ \wedge$
$((fst\ (ys!i))\vdash_c ((snd\ (ys!i))!(Suc\ j))\ \ \rightarrow ((snd\ (ys!i))!(Suc\ (Suc\ j))))\ \wedge$
$(\forall\, l<length\ ys.$
$l\neq i \longrightarrow (fst\ (ys!l))\vdash_c (snd\ (ys!l))!(Suc\ j)\ \ \rightarrow_e ((snd\ (ys!l))!(Suc\ (Suc\ j)))\ \ )))$
by *auto*
then have $(\ (\Gamma \vdash_p((snd\ \ (\Gamma,\ x\#xs))!(Suc\ j))\ \ \rightarrow ((snd\ \ (\Gamma,\ x\#xs))!(Suc\ (Suc\ j))))\ \wedge$
$(\exists\, i<length\ ys.$
$((fst\ (ys!i))\vdash_c ((snd\ (zs!i))!(\ j))\ \ \rightarrow ((snd\ (zs!i))!(\ (Suc\ j))))\ \wedge$
$(\forall\, l<length\ ys.$
$l\neq i \longrightarrow (fst\ (ys!l))\vdash_c (snd\ (zs!l))!(\ j)\ \ \rightarrow_e ((\ snd\ (zs!l))!(\ (Suc\ j)))\ )))$
proof –

have $f1$: $\Gamma \vdash_p$ snd $(\Gamma, x \# xs)$! $Suc\ j \rightarrow snd\ (\Gamma, x \# xs)$! $Suc\ (Suc\ j) \wedge i < length\ ys \wedge fst\ (ys\ !\ i) \vdash_c snd\ (ys\ !\ i)\ !\ Suc\ j \rightarrow snd\ (ys\ !\ i)\ !\ Suc\ (Suc\ j) \wedge (\forall n.\ (\neg\ n < length\ ys \vee n = i) \vee fst\ (ys\ !\ n) \vdash_c snd\ (ys\ !\ n)\ !\ Suc\ j \rightarrow_e snd\ (ys\ !\ n)\ !\ Suc\ (Suc\ j))$
using $f1$ **by** *blast*
have $f2$: $j < length\ (snd\ (\Gamma, xs))$
by (*meson Suc-lessD a11*)
have $f3$: $\forall n.\ \neg\ n < length\ zs \vee length\ (snd\ (zs\ !\ n)) = length\ (snd\ (\Gamma, xs))$
using *same-len same-length-def* **by** *blast*
have $\forall n.\ \neg\ n < length\ ys \vee snd\ (zs\ !\ n) = tl\ (snd\ (ys\ !\ n))$
using *tail* **by** *blast*
thus *?thesis*
using *f3 f2 f1* **by** (*metis* (*no-types*) *List.nth-tl a11 s-l-y-z*)
qed
then have($(\Gamma \vdash_p((snd\ (\Gamma, xs))!(j)) \rightarrow ((snd\ (\Gamma,xs))!((Suc\ j)))) \wedge$
$(\exists i<length\ zs.$
$((fst\ (zs!i))\vdash_c ((snd\ (zs!i))!(\ j)) \rightarrow ((snd\ (zs!i))!(\ (Suc\ j)))) \wedge$
$(\forall l<length\ zs.$
$l\neq i \longrightarrow (fst\ (zs!l))\vdash_c (snd\ (zs!l))!(\ j) \rightarrow_e ((\ snd\ (zs!l))!(\ (Suc\ j)))$
$)))$
using *same-env s-l-y-z fun-ys-eq-fun-zs* **by** *force*
then have($(fst\ (\Gamma, xs) \vdash_p((snd\ (\Gamma, xs))!(j)) \rightarrow ((snd\ (\Gamma,xs))!((Suc\ j)))) \wedge$
$(\exists i<length\ zs.$
$((fst\ (zs!i))\vdash_c ((snd\ (zs!i))!(\ j)) \rightarrow ((snd\ (zs!i))!(\ (Suc\ j)))) \wedge$
$(\forall l<length\ zs.$
$l\neq i \longrightarrow (fst\ (zs!l))\vdash_c (snd\ (zs!l))!(\ j) \rightarrow_e ((\ snd\ (zs!l))!(\ (Suc\ j)))$
$)))$
by *auto*
thus *?thesis*
by *auto*
next
assume $a22$:
$(\Gamma\vdash_p((snd\ (\Gamma, x\#xs))!(Suc\ j)) \rightarrow_e ((snd\ (\Gamma, x\#xs))!(Suc\ (Suc\ j)))) \wedge$
$(\forall i<length\ ys.\ (fst\ (ys!i))\vdash_c (snd\ (ys!i))!(Suc\ j) \rightarrow_e ((snd\ (ys!i))!(Suc\ (Suc\ j)))\ ))$
then have
$(\Gamma\vdash_p((snd\ (\Gamma, x\#xs))!(Suc\ j)) \rightarrow_e ((snd\ (\Gamma, x\#xs))!(Suc\ (Suc\ j)))) \wedge$
$(\forall i<length\ ys.\ (fst\ (ys!i))\vdash_c (snd\ (zs!i))!(j) \rightarrow_e ((snd\ (zs!i))!((Suc\ j)))$
$))$
using *Suc-j-l-ys tail s-l-y-z tl-zero-pair21* **by** *metis*
then have
$(\Gamma\vdash_p((snd\ (\Gamma, xs))!(j)) \rightarrow_e ((snd\ (\Gamma, xs))!((Suc\ j)))) \wedge$
$(\forall i<length\ zs.\ (fst\ (zs!i))\vdash_c (snd\ (zs!i))!(j) \rightarrow_e ((snd\ (zs!i))!((Suc\ j)))$
$))$
using *same-env s-l-y-z fun-ys-eq-fun-zs* **by** *fastforce*
thus *?thesis* **by** *auto*
qed

}
then have *compat-label* (Γ,*xs*) *zs*
using *compat-label-def* **by** *blast*
} **note** *same-label* = *this*
ultimately show *?thesis* **using** *conjoin-def* **by** *auto*
qed

lemma *clist-tail*:
assumes
a1:*length xs* = *length clist* **and**
a2: *ys* = (*map* (λ*i*. (Γ,(*fst i*,*s*)#*snd i*)) (*zip xs clist*))
shows ∀ *i* < *length ys*. *tl* (*snd* (*ys*!*i*)) = *clist*!*i*
using *a1 a2*
proof −
show *?thesis* **using** *a2*
by (*simp add*: *a1*)
qed

lemma *clist-map*:
assumes
a1:*length xs* = *length clist*
shows *clist* = *map* ((λ*p*. *tl* (*snd p*)) ∘ (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))) (*zip xs clist*)
proof −
have *f1*: *map snd* (*zip xs clist*) = *clist*
using *a1 map-snd-zip* **by** *blast*
have *map snd* (*zip xs clist*) = *map* ((λ*p*. *tl* (*snd p*)) ∘ (λ*p*. (Γ, (*fst p*, *s*) # *snd p*))) (*zip xs clist*)
by *simp*
thus *?thesis*
using *f1* **by** *presburger*
qed

lemma *clist-map1*:
assumes
a1:*length xs* = *length clist*
shows *clist* = *map* (λ*p*. *tl* (*snd p*)) (*map* (λ*i*. (Γ,(*fst i*,*s*)#*snd i*)) (*zip xs clist*))
proof −
have *clist* = *map* ((λ*p*. *tl* (*snd p*)) ∘ (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))) (*zip xs clist*)
using *a1 clist-map* **by** *fastforce*
thus *?thesis* **by** *auto*
qed

lemma *clist-map2*:
(*clist* = *map* (λ*p*. *tl* (*snd p*)) (*l*::(′*a* ×′*b list*) *list*)) ⟹
clist = *map* (λ*p*. (*snd p*)) (*map* (λ*p*. (*fst p*, *tl* (*snd p*))) (*l*::(′*a* ×′*b list*) *list*))

by *auto*

lemma *map-snd*:
  assumes *a1*: $y = map\ \ (\lambda x.\ f\ x)\ l$
  shows $y{=}(map\ snd\ (map\ (\lambda x.\ (g\ x,\ f\ x))\ l))$
by (*simp add*: *assms*)

lemmas *map-snd-sym* = *map-snd*[*THEN sym*]

lemma *map-snd'*:
  shows $map\ \ (\lambda x.\ f\ x)\ l{=}(map\ snd\ (map\ (\lambda x.\ (g\ x,\ f\ x))\ l))$
by *simp*

lemma *clist-snd*:
 assumes *a1*: $(\Gamma,\ a\ \#\ ys) \propto map\ (\lambda x.\ (fst\ x,\ tl\ (snd\ x)))$
   $(map\ (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))\ (zip\ xs\ clist))$ **and**
   a2: $length\ clist > 0 \wedge length\ clist = length\ xs$
 shows $clist = (map\ snd$
   $(map\ (\lambda x.\ (\Gamma,\ (fst\ x,\ snd\ (clist\ !\ 0\ !\ 0))\ \#\ snd\ x))$
   $(zip\ (map\ (\lambda x.\ fst\ (hd\ x))\ clist)\ (map\ tl\ clist))))$
proof −
  let $?concat\text{-}zip = (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))$
  let $?clist\text{-}ext = map\ ?concat\text{-}zip\ (zip\ xs\ clist)$
  let $?exec\text{-}run = (xs,\ s)\ \#\ a\ \#\ ys$
  let $?exec = (\Gamma,?exec\text{-}run)$
  let $?exec\text{-}ext = map\ (\lambda x.\ (fst\ x,\ tl\ (snd\ x)))\ ?clist\text{-}ext$
  let $?zip = (zip\ (map\ (\lambda x.\ fst\ (hd\ x))\ clist)$
   $(map\ (\lambda x.\ tl\ x)\ clist))$
 have Γ-*all*: $\forall\, i < length\ ?clist\text{-}ext.\ fst\ (?clist\text{-}ext\ !i) = \Gamma$
   by *auto*
 have *len*:$length\ xs = length\ clist$ **using** *a2* **by** *auto*
 then have *len-clist-exec*:
  $length\ clist = length\ ?exec\text{-}ext$
  by *fastforce*
 then have *len-clist-exec-map*:
  $length\ ?exec\text{-}ext =$
   $length\ (map\ (\lambda x.\ (\Gamma,\ (fst\ x{,}snd\ ((clist!0)!0))\#snd\ x))$
   $?zip)$
  by *fastforce*
 then have *clist-snd*:$clist = map\ (\lambda x.\ snd\ x)\ ?exec\text{-}ext$
  using *clist-map1* [*of xs clist* Γ *s*] *clist-map2 len* **by** *blast*
 then have *clist-len-eq-ays*:
   $\forall\, i < length\ clist.\ length(\ (clist!i)){=}length\ (snd\ (\Gamma{,}a\#ys))$
  using *len same-length-non-pair a1 conjoin-def*
  by *blast*
 then have *clist-gz*:$\forall\, i < length\ clist.\ length\ (clist!i) > 0$
  by *fastforce*
 have *clist-len-eq*:

$\forall i < length\ clist.\ \forall j < length\ clist.$
$length\ (clist\ !\ i) = length\ (clist\ !\ j)$
using *clist-len-eq-ays* **by** *auto*
have *clist-same-state*:
$\forall i < length\ clist.\ \forall j< length\ clist.\ \forall k<length\ \ (clist!i).$
$snd\ ((clist!i)!k) = snd\ ((clist!j)!k)$
proof –
have
$(\forall i <length\ clist.\ \forall j<length\ (snd\ (\Gamma,\ a\ \#\ ys)).\ snd((snd\ (\Gamma,\ a\ \#\ ys))!j) = snd(\ (clist!i)!j))$
using *len clist-snd conjoin-def a1 conjoin-def same-state-non-pair*
by *blast*
thus *?thesis* **using** *clist-len-eq-ays* **by** (*metis* (*no-types*))
qed
then have *clist-map*:
$clist = map\ (\lambda i.\ (fst\ i,snd\ ((clist!0)!0))\#snd\ i)\ ?zip$
using *list-as-map a2 clist-gz clist-len-eq* **by** *blast*
moreover have $map\ (\lambda i.\ (fst\ i,snd\ ((clist!0)!0))\#snd\ i)\ ?zip =$
$map\ snd\ (map\ (\lambda x.\ (\Gamma,\ (fst\ x,\ snd\ (clist\ !\ 0\ !\ 0))\ \#\ snd\ x))$
$(zip\ (map\ (\lambda x.\ fst\ (hd\ x))\ clist)\ (map\ tl\ clist)))$
using *map-snd′* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *list-as-zip*:
assumes *a1*: $(\Gamma,\ a\ \#\ ys) \propto map\ (\lambda x.\ (fst\ x,\ tl\ (snd\ x)))$
$(map\ (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))\ (zip\ xs\ clist))$ **and**
a2: $length\ clist > 0 \land length\ clist = length\ xs$
shows $map\ (\lambda x.\ (fst\ x,\ tl\ (snd\ x)))$
$(map\ (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))\ (zip\ xs\ clist)) =$
$map\ (\lambda x.\ (\Gamma,\ (fst\ x,snd\ ((clist!0)!0))\#snd\ x))$
$(zip\ (map\ (\lambda x.\ fst\ (hd\ x))\ clist)$
$(map\ (\lambda x.\ tl\ x)\ clist))$
proof –
let $?concat\text{-}zip = (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))$
let $?clist\text{-}ext = map\ ?concat\text{-}zip\ (zip\ xs\ clist)$
let $?exec\text{-}run = (xs,\ s)\ \#\ a\ \#\ ys$
let $?exec = (\Gamma,?exec\text{-}run)$
let $?exec\text{-}ext = map\ (\lambda x.\ (fst\ x,\ tl\ (snd\ x)))\ ?clist\text{-}ext$
let $?zip = (zip\ (map\ (\lambda x.\ fst\ (hd\ x))\ clist)$
$(map\ (\lambda x.\ tl\ x)\ clist))$
have Γ-*all*: $\forall i < length\ ?clist\text{-}ext.\ fst\ (?clist\text{-}ext\ !i) = \Gamma$
by *auto*
have *len*:$length\ xs = length\ clist$ **using** *a2* **by** *auto*
then have *len-clist-exec*:
$length\ clist = length\ ?exec\text{-}ext$
by *fastforce*
then have *len-clist-exec-map*:
$length\ ?exec\text{-}ext =$

```
          length (map (λx. (Γ, (fst x,snd ((clist!0)!0))#snd x))
                 ?zip)
  by fastforce
 then have clist-snd:clist = map (λx. snd x) ?exec-ext
   using clist-map1 [of xs clist Γ s] clist-map2 len by blast
 then have clist-len-eq-ays:
     ∀ i < length clist. length( (clist!i))=length (snd (Γ,a#ys))
   using len  same-length-non-pair a1 conjoin-def
   by blast
 then have clist-gz:∀ i < length clist. length (clist!i) > 0
   by fastforce
 have clist-len-eq:
    ∀ i < length clist. ∀ j < length clist.
       length (clist ! i) = length (clist ! j)
   using clist-len-eq-ays by auto
 have clist-same-state:
   ∀ i < length clist. ∀ j< length clist. ∀ k<length  (clist!i).
      snd ((clist!i)!k) = snd ((clist!j)!k)
 proof −
   have
     (∀ i <length clist. ∀ j<length (snd (Γ, a # ys)). snd((snd (Γ, a # ys))!j) =
snd( (clist!i)!j))
     using len clist-snd conjoin-def a1 conjoin-def same-state-non-pair
   by blast
   thus ?thesis using clist-len-eq-ays by (metis (no-types))
 qed
 then have clist-map:
   clist = map (λi. (fst i,snd ((clist!0)!0))#snd i) ?zip
   using list-as-map a2 clist-gz clist-len-eq by blast
 then have ∀ i < length clist.
             clist ! i = (fst (?zip!i),snd ((clist!0)!0)) # snd (?zip!i)
 using len nth-map length-map by (metis (no-types, lifting))
 then have
   ∀ i < length clist.
    ?exec-ext ! i = (Γ, (fst (?zip!i),snd ((clist!0)!0)) # snd (?zip!i))
 using Γ-all len  by fastforce
 moreover have ∀ i < length clist.
   (Γ, (fst (?zip!i),snd ((clist!0)!0)) # snd (?zip!i)) =
   (map (λx. (Γ, (fst x,snd ((clist!0)!0))#snd x))
                 ?zip)!i
 by auto
 ultimately have
    ∀ i < length clist.
      ?exec-ext ! i =(map (λx. (Γ, (fst x,snd ((clist!0)!0))#snd x))
                 ?zip)!i
 by auto
 then also have length clist = length ?exec-ext
 using len by fastforce
 ultimately have exec-ext-eq-clist-map:
```

$\forall\, i < length\ ?exec\text{-}ext.$
$?exec\text{-}ext\ !\ i = (map\ (\lambda x.\ (\Gamma,\ (fst\ x, snd\ ((clist!0)!0))\#snd\ x))$
$?zip)!i$
by *presburger*
then moreover have $length\ ?exec\text{-}ext =$
$length\ (map\ (\lambda x.\ (\Gamma,\ (fst\ x, snd\ ((clist!0)!0))\#snd\ x))$
$?zip)$
using *len clist-map* **by** *fastforce*
ultimately show *?thesis*
using *list-eq* **by** *blast*
qed

lemma *hd-nth*:
assumes $a1{:}i< length\ l \wedge (\ length(\ (l!i)) > 0)$
shows $f\ (hd\ (l!i)) = f\ (nth\ (l!i)\ 0)$
using *assms hd-conv-nth* **by** *fastforce*

lemma *map-hd-nth*:
assumes $a1{:}(\forall\, i < length\ l.\ length(\ (l!i)) > 0)$
shows $map\ (\lambda x.\ f\ (hd\ x))\ l = map\ (\lambda x.\ f\ (nth\ (x)\ 0))\ l$
proof −
have $\forall\, i < length\ l.\ (map\ (\lambda x.\ f\ (hd\ x))\ l)!i = f\ (nth\ (l!i)\ 0)$
using *hd-nth a1* **by** *auto*
moreover have $\forall\, i < length\ l.\ (map\ (\lambda x.\ f\ (nth\ x\ 0))\ l)!i = f\ (nth\ (l!i)\ 0)$
using *hd-nth a1* **by** *auto*
ultimately have $f1{:}\forall\, i < length\ l.\ (map\ (\lambda x.\ f\ (hd\ x))\ l)!i = (map\ (\lambda x.\ f\ (nth\ x\ 0))\ l)!i$
by *auto*
moreover have $f2{:}length\ (map\ (\lambda x.\ f\ (hd\ x))\ l) = length\ l$
by *auto*
moreover have $length\ (map\ (\lambda x.\ f\ (nth\ x\ 0))\ l) = length\ l$ **by** *auto*
ultimately show *?thesis* **using** *nth-equalityI* **by** *metis*
qed

lemma $i<length\ clist \Longrightarrow clist!i = (x1,ys) \Longrightarrow ys = (map\ (\lambda x.\ (fst\ (hd\ (snd\ x)),s)\#tl\ (snd\ x))\ clist)!i \Longrightarrow$
$ys = (map\ (\lambda x.\ (fst\ x,\ s)\#snd\ x)$
$(zip\ (map\ (\lambda x.\ fst\ (hd\ (snd\ x)))\ clist)$
$(map\ (\lambda x.\ tl\ (snd\ x))\ clist)))!i$
proof $(induct\ ys)$
case *Nil* **thus** *?case* **by** *auto*
next
case $(Cons\ y\ ys)$
have $\forall\, n\ ps\ f.\ \neg\ n < length\ ps \vee map\ f\ ps\ !\ n = (f\ (ps\ !\ n{::}'a \times ('b \times 'c)\ list){::}('b \times 'c)\ list)$
by *force*
hence $y\ \#\ ys = (fst\ (hd\ (snd\ (clist\ !\ i))),\ s)\ \#\ tl\ (snd\ (clist\ !\ i))$
using *Cons.prems(1) Cons.prems(3)* **by** *presburger*
thus *?case*

using *Cons.prems*(*1*) **by** *auto*
qed

lemma *clist-map-zip*:$xs \neq [] \Longrightarrow (\Gamma,(xs,s)\#ys) \propto clist \Longrightarrow$
$clist = map\ (\lambda i.\ (\Gamma,(fst\ i,s)\#(snd\ i)))\ (zip\ xs\ ((map\ (\lambda x.\ tl\ (snd\ x)))\ clist))$
proof −
let *?clist* = *map snd clist*
assume *a1*: $xs \neq []$
assume *a2*: $(\Gamma,(xs,s)\#ys) \propto clist$
then have *all-in-clist-not-empty*:$\forall\, i < length\ ?clist.\ (?clist!i) \neq []$
unfolding *conjoin-def same-length-def* **by** *auto*
then have *hd-clist*:$\forall\, i < length\ ?clist.\ hd\ (?clist!i) = (?clist!i)!0$
by (*simp add*: *hd-conv-nth*)
then have *all-xs*:$\forall\, i< length\ ?clist.\ fst\ (hd\ (?clist!i)) = xs!i$
using *a2* **unfolding** *conjoin-def same-program-def* **by** *auto*

then have *all-s*: $\forall\, i < length\ ?clist.\ snd\ (hd\ (?clist!i)) = s$
using *a2 hd-clist* **unfolding** *conjoin-def same-state-def* **by** *fastforce*
have *fst-clist-*Γ:$\forall\, i < length\ clist.\ fst\ (clist!i) = \Gamma$
using *a2* **unfolding** *conjoin-def same-functions-def* **by** *auto*
have *p2*:*length xs* = *length clist* **using** *conjoin-same-length a2*
by *fastforce*

then have $\forall\, i< length\ (map\ (\lambda x.\ fst\ (hd\ x))\ ?clist).$
$(map\ (\lambda x.\ fst\ (hd\ x))\ ?clist)!i = xs!i$
using *all-xs* **by** *auto*
also have $length\ (map\ (\lambda x.\ fst\ (hd\ x))\ ?clist) = length\ xs$ **using** *p2* **by** *auto*
ultimately have $(map\ (\lambda x.\ fst\ (hd\ x))\ ?clist) = xs$
using *nth-equalityI* **by** *metis*
then have *xs-clist*:$map\ (\lambda x.\ fst\ (hd\ (snd\ x)))\ clist = xs$ **by** *auto*

have *clist-hd-tl*:$\forall\, i < length\ ?clist.\ ?clist!i = hd\ (?clist!i)\ \#\ (tl\ (?clist!i))$
using *all-in-clist-not-empty list.exhaust-sel* **by** *blast*
then have $\forall\, i < length\ ?clist.\ ?clist!i =(fst\ (hd\ (?clist!i)),snd\ (hd\ (?clist!i)))\#$
$(tl\ (?clist!i))$
by *auto*
then have $?clist = map\ (\lambda x.\ (fst\ (hd\ x),snd\ (hd\ x))\#tl\ x)\ ?clist$
using *length-map list-eq-iff-nth-eq list-update-id map-update nth-list-update-eq*
by (*metis* (*no-types*, *lifting*) *length-map list-eq-iff-nth-eq list-update-id map-update nth-list-update-eq*)
then have $?clist = map\ (\lambda x.\ (fst\ (hd\ x),s)\#tl\ x)\ ?clist$
using *all-s length-map nth-equalityI nth-map*
by (*metis* (*no-types*, *lifting*))
then have *map-clist*:$map\ (\lambda x.\ (fst\ (hd\ (snd\ x)),s)\#tl\ (snd\ x))\ clist = ?clist$
by *auto*
then have $(map\ (\lambda x.\ (fst\ x,\ s)\#snd\ x)$

(*zip* (*map* (*λx*. *fst* (*hd* (*snd x*))) *clist*)
(*map* (*λx*. *tl* (*snd x*)) *clist*))) = *?clist*
using *map-clist* **by** (*auto intro*: *nth-equalityI*)
then have ∀ *i*<*length clist*. *clist*!*i* = (Γ,(*map* (*λx*. (*fst x*, *s*)#*snd x*)
(*zip xs*
(*map* (*λx*. *tl* (*snd x*)) *clist*)))!*i*)
using *xs-clist fst-clist-*Γ **by** *auto*
also have *length clist* = *length* (*map* (*λi*. (Γ,(*fst i*,*s*)#(*snd i*))) (*zip xs* ((*map* (*λx*. *tl* (*snd x*))) *clist*)))
using *p2* **by** *auto*
ultimately show *clist* = *map* (*λi*. (Γ,(*fst i*,*s*)#(*snd i*))) (*zip xs* ((*map* (*λx*. *tl* (*snd x*))) *clist*))
using *length-map length-zip nth-equalityI nth-map*
by (*metis* (*no-types*, *lifting*))
qed

lemma *aux-if'* :
assumes *a*:*length clist* > *0* ∧ *length clist* = *length xs* ∧
(∀ *i*<*length xs*. (Γ,(*xs*!*i*,*s*)#*clist*!*i*) ∈ *cptn*) ∧
((Γ,(*xs*, *s*)#*ys*) ∝ *map* (*λi*. (Γ,(*fst i*,*s*)#*snd i*)) (*zip xs clist*))
shows (Γ,(*xs*, *s*)#*ys*) ∈ *par-cptn*
using *a*
proof (*induct ys arbitrary*: *xs s clist*)
case *Nil* **then show** *?case* **by** (*simp add*: *par-cptn.ParCptnOne*)
next
case (*Cons a ys xs s clist*)
let *?concat-zip* = (*λi*. (Γ, (*fst i*, *s*) # *snd i*))
let *?com-clist-xs* = *map ?concat-zip* (*zip xs clist*)
let *?xs-a-ys-run* = (*xs*, *s*) # *a* # *ys*
let *?xs-a-ys-run-exec* = (Γ,*?xs-a-ys-run*)
let *?com-clist'* = *map* (*λx*. (*fst x*, *tl* (*snd x*))) *?com-clist-xs*
let *?xs'* = (*map* (*λx*. *fst* (*hd x*)) *clist*)
let *?clist'* = (*map* (*λx*. *tl x*) *clist*)
let *?zip-xs'-clist'* = *zip ?xs'*
?clist'
obtain *as sa* **where** *a-pair*:*a*=(*as*,*sa*) **by** *fastforce*
let *?comp-clist'-alt* = *map* (*λx*. (Γ, (*fst x*,*snd* ((*clist*!*0*)!*0*))#*snd x*)) *?zip-xs'-clist'*

let *?clist'-alt* = *map* (*λx*. *snd x*) *?comp-clist'-alt*
let *?comp-a-ys* = (Γ, (*as*,*sa*) # *ys*)
have *conjoin-hyp1*:
(Γ, (*as*,*sa*) # *ys*) ∝ *?com-clist'*
using *conjoin-tl* **using** *a-pair Cons* **by** *blast*
then have *conjoin-hyp*:
(Γ, (*as*,*sa*) # *ys*) ∝ *map* (*λx*. (Γ, (*fst x*,*snd* ((*clist*!*0*)!*0*))#*snd x*)) *?zip-xs'-clist'*
using *list-as-zip Cons.prems* **by** *fastforce*
have *len*:*length xs* = *length clist* **using** *Cons* **by** *auto*
have *clist-snd-map*:
(*map snd*

(*map* (λx. (Γ, (*fst x*, *snd* (*clist* ! *0* ! *0*)) # *snd x*))
(*zip* (*map* (λx. *fst* (*hd x*)) *clist*) (*map tl clist*)))) = *clist*
using *clist-snd Cons.prems conjoin-hyp1* **by** *fastforce*
have *eq-len-clist-clist'*:
length ?clist' > *0* **using** *Cons.prems* **by** *auto*
have ($\forall$ *i* <*length clist*. $\forall$ *j*<*length* (*snd ?comp-a-ys*). *snd*((*snd ?comp-a-ys*)!*j*) = *snd*((*clist*!*i*)!*j*))
using *clist-snd-map conjoin-hyp conjoin-def same-state-non-pair*[*of ?comp-a-ys ?comp-clist'-alt ?clist'-alt*]
by *fastforce*
then have $\forall$ *i*<*length clist*.
sa = *snd* ((*clist* ! *i*)!*0*) **by** *fastforce*
also have *clist-i-grz*:($\forall$ *i* <*length clist*. *length*((*clist*!*i*)) > *0*)
using *clist-snd-map conjoin-hyp conjoin-def same-length-non-pair*[*of ?comp-a-ys ?comp-clist'-alt ?clist'-alt*]
by *fastforce*
ultimately have *all-i-sa-hd-clist*:$\forall$ *i*<*length clist*.
sa = *snd* (*hd* (*clist* ! *i*))
by (*simp add*: *hd-conv-nth*)
have *as-sa-eq-xs'-s'*:*as* = *?xs'* $\wedge$ *sa* = *snd* ((*clist*!*0*)!*0*)
proof −
have ($\forall$ *j*<*length* (*snd ?comp-a-ys*). *fst*((*snd ?comp-a-ys*)!*j*) =
map (λx. *fst*(*nth x j*)) *?clist'-alt*)
using *conjoin-hyp conjoin-def same-program-non-pair*[*of ?comp-a-ys ?comp-clist'-alt ?clist'-alt*]
by *fast*
then have *are-eq*:*fst*((*snd ?comp-a-ys*)!*0*) =
map (λx. *fst*(*nth x 0*)) *?clist'-alt* **by** *fastforce*
have *fst-exec-is-as*:*fst*((*snd ?comp-a-ys*)!*0*) =*as* **by** *auto*
then have *map* (λx. *fst*(*hd x*)) *clist*=*map* (λx. *fst*(*x*!*0*)) *clist*
using *map-hd-nth clist-i-grz* **by** *auto*
then have *map* (λx. *fst*(*nth x 0*)) *?clist'-alt* =*?xs'* **using** *clist-snd-map map-hd-nth*
by *fastforce*
moreover have ($\forall$ *i* <*length clist*. $\forall$ *j*<*length* (*snd ?comp-a-ys*). *snd*((*snd ?comp-a-ys*)!*j*) = *snd*((*clist*!*i*)!*j*))
using *clist-snd-map conjoin-hyp conjoin-def same-state-non-pair*[*of ?comp-a-ys ?comp-clist'-alt ?clist'-alt*]
by *fastforce*
ultimately show *?thesis* **using** *are-eq fst-exec-is-as*
using *Cons.prems* **by** *force*
qed
then have *conjoin-hyp*:
(Γ, (*as*,*sa*) # *ys*) $\propto$ *map* (λx. (Γ, (*fst x*,*sa*)#*snd x*))
(*zip as* (*map tl clist*))
using *conjoin-hyp* **by** *auto*
then have *eq-len-as-clist'*:
length as = *length ?clist'* **using** *Cons.prems as-sa-eq-xs'-s'* **by** *auto*
then have *len-as-ys-eq*:*length as* = *length xs* **using** *Cons.prems* **by** *auto*

have $(\forall\, i<length\ as.\ (\Gamma,\ ((as!i),sa)\#(map\ (\lambda x.\ tl\ x)\ clist)!i) \in cptn)$
using *Cons.prems cptn-dest clist-snd-map len*
proof −
have $\forall\, i<length\ clist.\ clist!i = (hd\ (clist!i))\#(tl\ (clist!i))$
using *clist-i-grz*
by *auto*
then have $(\forall\, i<length\ clist.\ (\Gamma,\ (xs\ !\ i,\ s)\ \#\ (hd\ (clist!i))\#(tl\ (clist!i))) \in cptn)$
using *Cons.prems* **by** *auto*
then have $f1{:}(\forall\, i<length\ clist.\ (\Gamma,\ (hd\ (clist!i))\#(tl\ (clist!i))) \in cptn)$
by (*metis list.distinct*(*2*) *tl-in-cptn*)
then have $(\forall\, i<length\ clist.\ (\Gamma,\ ((as!i),sa)\#(tl\ (clist!i))) \in cptn)$
using *as-sa-eq-xs′-s′ all-i-sa-hd-clist* **by** *auto*
then have $(\forall\, i<length\ clist.\ (\Gamma,\ ((as!i),sa)\#(map\ (\lambda x.\ tl\ x)\ clist)!i) \in cptn)$
by *auto*
thus *?thesis* **using** *len clist-i-grz len-as-ys-eq* **by** *auto*
qed
then have *a-ys-par-cptn*:$(\Gamma,\ (as,\ sa)\ \#\ ys) \in par\text{-}cptn$
using
conjoin-hyp eq-len-clist-clist′ eq-len-as-clist′[*THEN sym*] *Cons.hyps*
by *blast*
have Γ-*all*: $\forall\, i < length\ ?com\text{-}clist\text{-}xs.\ fst\ (?com\text{-}clist\text{-}xs\ !i) = \Gamma$
by *auto*
have *Gamma*: $\Gamma = (fst\ ?xs\text{-}a\text{-}ys\text{-}run\text{-}exec)$ **by** *fastforce*
have *exec*: $?xs\text{-}a\text{-}ys\text{-}run = (snd\ ?xs\text{-}a\text{-}ys\text{-}run\text{-}exec)$ **by** *fastforce*
have *split-par*:
$\Gamma\vdash_p ((xs,\ s)\ \#\ a\ \#\ ys)\ !\ 0 \rightarrow ((a\ \#\ ys)\ !\ 0) \lor$
$\Gamma\vdash_p ((xs,\ s)\ \#\ a\ \#\ ys)\ !\ 0 \rightarrow_e ((a\ \#\ ys)\ !\ 0)$
using *compat-label-def compat-label-tran-0*
Cons.prems Gamma exec
compat-label-tran-0[*of* $(\Gamma,\ (xs,\ s)\ \#\ a\ \#\ ys)$
$(map\ (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))\ (zip\ xs\ clist))$]
unfolding *conjoin-def* **by** *auto*
{
assume $\Gamma\vdash_p ((xs,\ s)\ \#\ a\ \#\ ys)\ !\ 0 \rightarrow ((a\ \#\ ys)\ !\ 0)$
then have $(\Gamma,\ (xs,\ s)\ \#\ a\ \#\ ys) \in par\text{-}cptn$
using *a-ys-par-cptn a-pair par-cptn.ParCptnComp* **by** *fastforce*
} **note** *env-sol=this*
{
assume $\Gamma\vdash_p ((xs,\ s)\ \#\ a\ \#\ ys)\ !\ 0 \rightarrow_e ((a\ \#\ ys)\ !\ 0)$
then have *env-tran*: $\Gamma\vdash_p (xs,\ s) \rightarrow_e (as,sa)$ **using** *a-pair* **by** *auto*
have $xs = as$
by (*meson env-pe-c-c′-false env-tran*)
then have $(\Gamma,\ (xs,\ s)\ \#\ a\ \#\ ys) \in par\text{-}cptn$
using *a-ys-par-cptn a-pair env-tran ParCptnEnv* **by** *blast*
}
then show $(\Gamma,\ (xs,\ s)\ \#\ a\ \#\ ys) \in par\text{-}cptn$ **using** *env-sol Cons split-par* **by** *fastforce*
qed

lemma *mapzip-upd*: *length as* = *length clist* $\Longrightarrow$
(*map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*]) =
map (λj. ((*fst j*, *sa*)#*snd j*)) (*zip as clist*)
proof −
assume *a2*: *length as* = *length clist*
have $\forall$ *i* < *length* (*map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*]). (*map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*])!*i* = *map* (λj. ((*fst j*, *sa*)#*snd j*)) (*zip as clist*)!*i*
using *a2*
by *auto*
moreover have *length* (*map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*]) =
length (*map* (λj. ((*fst j*, *sa*)#*snd j*)) (*zip as clist*))
using *a2* **by** *auto*
ultimately have (*map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*]) = *map* (λj. ((*fst j*, *sa*)#*snd j*)) (*zip as clist*)
using *nth-equalityI* **by** *blast*
thus *map* (λj. (*as* ! *j*, *sa*) # *clist* ! *j*) [*0*..<*length as*] =
map (λj. (*fst j*, *sa*) # *snd j*) (*zip as clist*)
by *auto*
qed

lemma *aux-if* :
assumes *a*: *length clist* = *length xs* $\wedge$
($\forall$ *i*<*length xs*. (Γ,(*xs*!*i*,*s*)#*clist*!*i*) $\in$ *cptn*) $\wedge$
((Γ,(*xs*, *s*)#*ys*) $\propto$ *map* (λi. (Γ,(*fst i*,*s*)#*snd i*)) (*zip xs clist*))
shows (Γ,(*xs*, *s*)#*ys*) $\in$ *par-cptn*
using *a*
proof (*cases length clist*)
case *0*
then have *clist-empty*:*clist* = [] **by** *auto*
then have *map-clist-empty*:*map* (λi. (Γ,(*fst i*,*s*)#*snd i*)) (*zip xs clist*) = []
by *fastforce*
then have *conjoin*:(Γ,(*xs*, *s*)#*ys*) $\propto$ [] **using** *a* **by** *auto*
then have *all-eq*:$\forall$ *j*<*length* (*snd* (Γ,(*xs*, *s*)#*ys*)). *fst* (*snd* (Γ,(*xs*, *s*)#*ys*) ! *j*) = []
using *conjoin-def same-program-def*
by (*simp add*: *conjoin-def same-program-def*)
from *conjoin*
show *?thesis* **using** *conjoin*
proof (*induct ys arbitrary*: *s xs*)
case *Nil* **then show** *?case* **by** (*simp add*: *par-cptn.ParCptnOne*)
next
case (*Cons a ys*)
then have *conjoin-ind*:(Γ, (*xs*, *s*) # *a* # *ys*) $\propto$ [] **by** *auto*
then have (Γ,(*a* # *ys*)) $\propto$ []
by (*auto simp add*:*conjoin-def same-length-def*
same-state-def same-program-def same-functions-def
compat-label-def)

moreover obtain *as sa* **where** *pair-a*: $a=(as,sa)$ **using** *Cons* **by** *fastforce*
ultimately have *ays-par-cptn*:$(\Gamma,\ a\ \#\ ys) \in par\text{-}cptn$ **using** *Cons.hyps* **by** *auto*
have $\forall j.\ Suc\ j<length\ (snd\ (\Gamma,(xs,\ s)\#(as,sa)\#ys)) \longrightarrow$
$\neg(\exists\, i<length\ [].$
$((fst\ ([]!i))\vdash_c\ ((snd\ ([]!i))!j)\ \ \rightarrow ((snd\ ([]!i))!(Suc\ j))))$
using *conjoin-def compat-label-def* **by** *fastforce*
then have $(\forall j.\ Suc\ j<length\ (snd\ (\Gamma,(xs,\ s)\#(as,sa)\#ys)) \longrightarrow$
$((fst\ (\Gamma,(xs,\ s)\#(as,sa)\#ys))\vdash_p((snd\ (\Gamma,(xs,\ s)\#(as,sa)\#ys))!j)$
$\rightarrow_e\ ((snd\ (\Gamma,(xs,\ s)\#(as,sa)\#ys))!(Suc\ j))))$
using *conjoin-def compat-label-def conjoin-ind pair-a* **by** *blast*
then have *env-tran*:$\Gamma\vdash_p\ (xs,\ s)\ \ \rightarrow_e\ (as,sa)$ **by** *auto*
then show $(\Gamma,\ (xs,\ s)\ \#\ a\ \#\ ys) \in par\text{-}cptn$
using *ays-par-cptn pair-a env-tran ParCptnEnv env-pe-c-c'-false* **by** *blast*
qed
next
case *Suc*
then have *length clist* > 0 **by** *auto*
then show *?thesis* **using** *a aux-if'* **by** *blast*
qed

lemma *snormal-enviroment*:$s = Normal\ nsa \lor s = sa \land (\forall sa.\ s \neq Normal\ sa) \Longrightarrow$
$\Gamma\vdash_c\ (x,\ s) \rightarrow_e\ (x,\ sa)$
by (*metis Env Env-n*)

lemma *aux-onlyif* [*rule-format*]: $\forall xs\ s.\ (\Gamma,(xs,\ s)\#ys) \in par\text{-}cptn \longrightarrow$
$(\exists\, clist.\ (length\ clist = length\ xs) \land$
$(\Gamma,\ (xs,\ s)\#ys) \propto map\ (\lambda i.\ (\Gamma,\ (fst\ i,s)\#(snd\ i)))\ (zip\ xs\ clist) \land$
$(\forall\, i<length\ xs.\ (\Gamma,\ (xs!i,s)\#(clist!i)) \in cptn))$
proof (*induct ys*)
case *Nil*
{**fix** *xs s*
assume $(\Gamma,\ [(xs,\ s)]) \in par\text{-}cptn$
have *f1*:$length\ (map\ (\lambda i.\ [])\ [0..<length\ xs]) = length\ xs$ **by** *auto*
have *f2*:$(\Gamma,\ [(xs,\ s)]) \propto map\ (\lambda i.\ (\Gamma,\ (fst\ i,\ s)\ \#\ snd\ i))$
$(zip\ xs\ (map\ (\lambda i.\ [])\ [0..<length\ xs]))$
unfolding *conjoin-def same-length-def same-functions-def same-state-def same-program-def compat-label-def*
by(*simp*, *rule nth-equalityI*,*simp*,*simp*)
note $h = conjI[OF\ f1\ f2]$
have *f3*:$(\forall\, i<length\ xs.\ (\Gamma,\ (xs\ !\ i,\ s)\ \#\ (map\ (\lambda i.\ [])\ [0..<length\ xs])\ !\ i) \in cptn)$
by (*simp add*: *cptn.CptnOne*)
note $this = conjI[OF\ h\ f3]$
}
thus *?case* **by** *blast*
next
case (*Cons a ys*)

{**fix** *xs s*
assume *a1*:(Γ, (*xs*, *s*) # *a* # *ys*) ∈ *par-cptn*
then obtain *as sa* **where** *a-pair*: *a*=(*as*,*sa*) **by** *fastforce*
then have *par-cptn'*:(Γ,((*as*,*sa*)#*ys*)) ∈ *par-cptn*
using *a1 par-cptn-dest* **by** *blast*
then obtain *clist* **where** *hyp*:
length clist = *length as* ∧
(Γ, (*as*, *sa*) #
ys) ∝ *map* (λ*i*. (Γ, (*fst i*, *sa*) # *snd i*)) (*zip as clist*) ∧
(∀ *i*<*length as*. (Γ, (*as* ! *i*, *sa*) # *clist* ! *i*) ∈ *cptn*)
using *Cons.hyps* **by** *fastforce*
have *a11*:(Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*) ∈ *par-cptn* **using** *a1 a-pair* **by** *auto*
have *par-cptn-dest*:Γ$\vdash_p$ (*xs*, *s*) $\rightarrow_e$ (*as*, *sa*) ∨ Γ$\vdash_p$ (*xs*, *s*) → (*as*, *sa*)
using *par-cptn-elim-cases par-cptn' a1 a-pair* **by** *blast*
{
assume *a1*: Γ$\vdash_p$ (*xs*, *s*) $\rightarrow_e$ (*as*, *sa*)
then have *xs-as-eq*:*xs*=*as* **by** (*meson env-pe-c-c'-false*)
then have *ce*:∀ *i* < *length xs*. Γ$\vdash_c$ (*xs*!*i*, *s*) $\rightarrow_e$ (*as*!*i*, *sa*) **using** *a1 pe-ce* **by** *fastforce*
let *?clist*=(*map* (λ*j*. (*xs*!*j*, *sa*)#(*clist*!*j*)) [*0*..<*length xs*])
have *s1*:*length ?clist* = *length xs*
by *auto*
have *s2*:(∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *?clist* ! *i*) ∈ *cptn*)
using *a1 hyp CptnEnv xs-as-eq ce* **by** *fastforce*
have *s3*:(Γ, (*xs*, *s*) #
(*as*,*sa*) # *ys*) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*)
proof −
have *s-len*:*same-length* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp conjoin-def same-length-def xs-as-eq a1* **by** *fastforce*
have *s-state*: *same-state* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp*
apply (*simp add*:*hyp conjoin-def same-state-def a1*)
apply *clarify*
apply(*case-tac j*)
by (*simp add*: *xs-as-eq*,*simp add*: *xs-as-eq*)
have *s-function*: *same-functions* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp conjoin-def same-functions-def a1* **by** *fastforce*
have *s-program*: *same-program* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp*
apply (*simp add*:*hyp conjoin-def same-program-def same-length-def a1*)

```
          apply clarify
          apply(case-tac j)
            apply(rule nth-equalityI)
            apply(simp,simp)
          by(rule nth-equalityI, simp add: hyp xs-as-eq, simp add:xs-as-eq)
      have s-compat:compat-label (Γ, (xs, s) # (xs,sa) # ys)
                  (map (λi. (Γ, (fst i, s) # snd i))
                          (zip xs ?clist))
        using hyp a1 pe-ce
        apply (simp add:hyp conjoin-def compat-label-def)
        apply clarify
        apply(case-tac j,simp add: xs-as-eq)
           apply blast
          apply (simp add: xs-as-eq step-e.intros step-pe.intros)
         apply clarify
        apply(erule-tac x=nat in allE,erule impE,assumption)
        apply(erule disjE,simp)
        apply clarify
        apply(rule-tac x=i in exI)
        using hyp by (fastforce)+
     thus ?thesis using s-len s-program s-state s-function conjoin-def xs-as-eq
        by blast
  qed
  then have
   (∃ clist.
            length clist = length xs ∧
            (Γ, (xs, s) #
                  a # ys) ∝ map (λi. (Γ, (fst i, s) # snd i))
                          (zip xs clist) ∧
            (∀ i<length xs. (Γ, (xs ! i, s) # clist ! i) ∈ cptn))
  using s1 s2 a-pair by blast
} note s1=this

{
  assume a1':Γ⊢p (xs, s) → (as, sa)
  then obtain i r where
    inter-tran:i < length xs ∧ Γ⊢c (xs ! i, s) → (r, sa) ∧ as = xs[i := r]
  using step-p-pair-elim-cases by metis
  then have xs-as-eq-len: length xs = length as by simp
  from inter-tran
   have s-states:∃ nsa. s=Normal nsa ∨ (s=sa ∧ (∀ sa. (s≠Normal sa)))
   using step-not-normal-s-eq-t by blast
  have as-xs:∀ i'<length as. (i'=i ∧ as!i'=r) ∨ (as!i'=xs!i')
    using xs-as-eq-len by (simp add: inter-tran nth-list-update)
 let ?clist=(map (λj. (as!j, sa)#(clist!j)) [0..<length xs]) [i:=((r, sa)#(clist!i))]
  have s1:length ?clist = length xs
    by auto
  have s2:(∀ i'<length xs. (Γ, (xs ! i', s) # ?clist ! i') ∈ cptn)
     proof -
```

```
    {fix i′
     assume a1:i′ < length xs
     have (Γ, (xs ! i′, s) # ?clist ! i′) ∈ cptn
     proof (cases i=i′)
       case True
        thus ?thesis using inter-tran  hyp cptn.CptnComp
         apply simp
         by fastforce
     next
       case False
       thus ?thesis using s-states inter-tran  False hyp cptn.CptnComp a1
         apply clarify
         apply simp
         apply(erule-tac x=i′ in allE)
         apply (simp)
         apply(rule CptnEnv)
         by (auto simp add: Env Env-n)
     qed
    }
   thus ?thesis by fastforce
 qed
then have s3:(Γ, (xs, s) #
                 (as,sa) # ys) ∝ map (λi. (Γ, (fst i, s) # snd i))
                             (zip xs ?clist)
proof −
   from hyp have
    len-list:length clist = length as by auto
   from hyp have same-len:same-length (Γ, (as, sa) # ys)
               (map (λi. (Γ, (fst i, sa) # snd i)) (zip as clist))
     using conjoin-def by auto
   have s-len: same-length (Γ, (xs, s) # (as,sa) # ys)
                    (map (λi. (Γ, (fst i, s) # snd i))
                             (zip xs ?clist))
     using
       same-len  inter-tran
       unfolding conjoin-def same-length-def
       apply clarify
       apply(case-tac i=ia)
       by (auto simp add: len-list)
   have s-state: same-state (Γ, (xs, s) # (as,sa) # ys)
                    (map (λi. (Γ, (fst i, s) # snd i))
                             (zip xs ?clist))
         using hyp inter-tran unfolding conjoin-def same-state-def
          apply clarify
          apply(case-tac j, simp, simp (no-asm-simp))
          apply(case-tac i=ia,simp , simp )
           by (metis (no-types, hide-lams) as-xs nth-list-update-eq xs-as-eq-len)

   have s-function: same-functions (Γ, (xs, s) # (as,sa) # ys)
```

(*map* ($\lambda i.$ (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp conjoin-def same-functions-def a1* **by** *fastforce*
have *s-program*: *same-program* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* ($\lambda i.$ (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *hyp inter-tran* **unfolding** *conjoin-def same-program-def*
apply *clarify*
apply(*case-tac j*,*simp*)
apply(*rule nth-equalityI*,*simp*,*simp*)
apply *simp*
apply(*rule nth-equalityI*,*simp*,*simp*)
apply(*erule-tac x=nat* **and** $P=\lambda j.\ H\ j \longrightarrow (fst\ (a\ j))=((b\ j))$ **for** *H a b* **in** *allE*)
apply(*case-tac nat*)
apply *clarify*
apply(*case-tac i=ia*,*simp*,*simp*)
apply *clarify*
by(*case-tac i=ia*,*simp*,*simp*)
have *s-compat*:*compat-label* (Γ, (*xs*, *s*) # (*as*,*sa*) # *ys*)
(*map* ($\lambda i.$ (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs ?clist*))
using *inter-tran hyp s-states*
unfolding *conjoin-def compat-label-def*
apply *clarify*
apply(*case-tac j*)
apply(*rule conjI*,*simp*)
apply(*erule ParComp*,*assumption*)
apply *clarify*
apply(*rule exI*[**where** *x=i*],*simp*)
apply *clarify*
apply (*rule snormal-enviroment*,*assumption*)
apply *simp*
apply(*erule-tac x=nat* **and** $P=\lambda j.\ H\ j \longrightarrow (P\ j \vee Q\ j)$ **for** *H P Q* **in** *allE*,*simp*)
apply (*thin-tac* $s = Normal\ nsa \vee s = sa \wedge (\forall sa.\ s \neq Normal\ sa)$)
apply(*erule disjE*)
apply *clarify*
apply(*rule-tac x=ia* **in** *exI*,*simp*)
apply(*rule conjI*)
apply(*case-tac i=ia*,*simp*,*simp*)
apply *clarify*
apply(*case-tac i=l*,*simp*)
apply(*case-tac l=ia*,*simp*,*simp*)
apply(*erule-tac x=l* **in** *allE*,*erule impE*,*assumption*,*erule impE*, *assumption*,*simp*)
apply *simp*
apply(*erule-tac x=l* **in** *allE*,*erule impE*,*assumption*,*erule impE*, *assumption*,*simp*)

apply *clarify*
apply (*thin-tac* $\forall$ *ia*<*length xs*. (Γ, (*xs*[*i* := *r*] ! *ia*, *sa*) # *clist* ! *ia*) ∈ *cptn*)
apply(*erule-tac x*=*ia* **and** *P*=λ*j*. *H j* ⟶ (*P j*) **for** *H P* **in** *allE*, *erule impE*, *assumption*)
by(*case-tac i*=*ia*,*simp*,*simp*)
thus *?thesis* **using** *s-len s-program s-state s-function conjoin-def*
by *blast*
qed
then have (∃ *clist*.
length clist = *length xs* ∧
(Γ, (*xs*, *s*) #
a # *ys*) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs clist*) ∧
(∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*))
using *s1 s2 a-pair* **by** *blast*
}
then have
(∃ *clist*.
length clist = *length xs* ∧
(Γ, (*xs*, *s*) #
a # *ys*) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs clist*) ∧
(∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*))
using *s1 par-cptn-dest* **by** *fastforce*
}
thus *?case* **by** *auto*
qed

lemma *one-iff-aux-if*:*xs*≠[] ⟹ (∀ *ys*. ((Γ,((*xs*, *s*)#*ys*)) ∈ *par-cptn*) =
(∃ *clist*. *length clist*= *length xs* ∧
((Γ,(*xs*, *s*)#*ys*) ∝ *map* (λ*i*. (Γ,(*fst i*,*s*)#(*snd i*))) (*zip xs clist*)) ∧
(∀ *i*<*length xs*. (Γ,(*xs*!*i*,*s*)#(*clist*!*i*)) ∈ *cptn*))) ⟹
(*par-cp* Γ (*xs*) *s* = {(Γ*1*,*c*). ∃ *clist*. (*length clist*)=(*length xs*) ∧
(∀ *i*<*length clist*. *clist*!*i* ∈ *cp* Γ (*xs*!*i*) *s*) ∧ (Γ,*c*) ∝ *clist* ∧ Γ*1*=Γ})
proof
assume *a1*:*xs*≠[]
assume *a2*:∀ *ys*. ((Γ, (*xs*, *s*) # *ys*) ∈ *par-cptn*) =
(∃ *clist*.
length clist = *length xs* ∧
(Γ,
(*xs*, *s*) #
ys) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
(*zip xs clist*) ∧
(∀ *i*<*length xs*.
(Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*))
show *par-cp* Γ *xs s* ⊆
{(Γ*1*, *c*). ∃ *clist*.
length clist = *length xs* ∧
(∀ *i*<*length clist*. *clist* ! *i* ∈ *cp* Γ (*xs* ! *i*) *s*) ∧

```
            (Γ, c) ∝ clist ∧ Γ1 = Γ}
proof−{
  fix x
  let ?show = x∈ {(Γ1, c). ∃clist.
    length clist = length xs ∧
    (∀i<length clist. clist ! i ∈ cp Γ (xs ! i) s) ∧
     (Γ, c) ∝ clist ∧ Γ1 = Γ}
  assume a3:x∈par-cp Γ xs s
  then obtain y where x-pair: x=(Γ,y)
    unfolding par-cp-def by auto
  have ?show
  proof (cases y)
     case Nil then
      show ?show using a1 a2 a3 x-pair
       unfolding par-cp-def cp-def
       by (force elim:par-cptn.cases)
  next
     case (Cons a list) then
       show ?show using a1 a2 a3 x-pair
       unfolding par-cp-def cp-def
          by(auto, rule-tac x=map (λi. (Γ,(fst i, s) # snd i)) (zip xs clist) in
exI,simp)
  qed
} thus ?thesis using a1 a2 by auto
qed
{
show {(Γ1, c). ∃clist.
       length clist = length xs ∧
       (∀i<length clist. clist ! i ∈ cp Γ (xs ! i) s) ∧
       (Γ, c) ∝ clist ∧ Γ1 = Γ} ⊆ par-cp Γ xs s using a1 a2
proof−
  {
  fix x
  assume a3:x∈{(Γ1, c). ∃clist.
       length clist = length xs ∧
       (∀i<length clist. clist ! i ∈ cp Γ (xs ! i) s) ∧
       (Γ, c) ∝ clist ∧ Γ1 = Γ}
  then obtain c where x-pair: x=(Γ,c)  by auto
  then obtain clist where
   props:length clist = length xs ∧
        (∀i<length clist. clist ! i ∈ cp Γ (xs ! i) s) ∧
        (Γ, c) ∝ clist  using a3 by auto
  then have x∈par-cp Γ xs s
    proof (cases c)
      case Nil
      have clist-0:
        clist ! 0 ∈ cp Γ (xs ! 0) s using props a1
      by auto
      thus x∈par-cp Γ xs s
```

using *a1 a2 props Nil x-pair*
unfolding *cp-def conjoin-def same-length-def*
apply *clarify*
by(*erule cptn.cases*,*fastforce*,*fastforce*,*fastforce*)
next
case (*Cons a ys*)
then obtain *a1 a2* **where** *a-pair*: *a*=(*a1*,*a2*)
using *props* **by** *fastforce*
from *a2* **have**
a2:(((Γ, (*xs*, *s*) # *ys*) $\in$ *par-cptn*) =
($\exists$ *clist*. *length clist* = *length xs* $\wedge$
(Γ, (*xs*, *s*) # *ys*) $\propto$ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*)) (*zip xs clist*) $\wedge$
($\forall$ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) $\in$ *cptn*))) **by** *auto*
have *a2-s*:*a2*=*s* **using** *a1 props a-pair Cons*
unfolding *conjoin-def same-state-def cp-def*
by *force*
have *a1-xs*:*a1* = *xs*
using *props a-pair Cons*
unfolding *par-cp-def conjoin-def same-program-def cp-def*
apply *clarify*
apply(*erule-tac x=0* **and** *P*=λ*j*. *H j* $\longrightarrow$ (*fst* (*s j*))=((*t j*)) **for** *H s t* **in** *allE*)
by(*rule nth-equalityI*,*auto*)
then have *conjoin-clist-xs*:(Γ, (*xs*,*s*)#*ys*) $\propto$ *clist*
using *a1 props a-pair Cons a1-xs a2-s* **by** *auto*
also then have *clist* = *map* (λ*i*. (Γ,(*fst i*,*s*)#(*snd i*))) (*zip xs* ((*map* (λ*x*. *tl* (*snd x*))) *clist*))
using *clist-map-zip a1* **by** *fastforce*
ultimately have *conjoin-map*:(Γ, (*xs*, *s*) # *ys*) $\propto$ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*)) (*zip xs* ((*map* (λ*x*. *tl* (*snd x*))) *clist*))
using *props x-pair Cons a-pair a1-xs a2-s* **by** *auto*
have $\bigwedge$*n*. $\neg$ *n* < *length xs* $\vee$ *clist* ! *n* $\in$ {(*f*, *ps*). *ps* ! *0* = (*xs* ! *n*, *a2*) $\wedge$ (Γ, *ps*) $\in$ *cptn* $\wedge$ *f* = Γ}
using *a1-xs a2-s props cp-def* **by** *fastforce*
then have *clist-cptn*:($\forall$ *i*<*length clist*. (*fst* (*clist*!*i*) = Γ) $\wedge$
(Γ, *snd* (*clist*!*i*)) $\in$ *cptn* $\wedge$
(*snd* (*clist*!*i*))!*0* = (*xs*!*i*,*s*))
using *a1-xs a2-s props* **by** *fastforce*

{**fix** *i*
assume *a4*: *i*<*length xs*
then have *clist-i-cptn*:(*fst* (*clist*!*i*) = Γ) $\wedge$
(Γ, *snd* (*clist*!*i*)) $\in$ *cptn* $\wedge$
(*snd* (*clist*!*i*))!*0* = (*xs*!*i*,*s*)
using *props clist-cptn* **by** *fastforce*
from *a4 props* **have** *a4'*:*i*<*length clist* **by** *auto*
have *lengz*:*length* (*snd* (*clist*!*i*))>*0*
using *conjoin-clist-xs a4'*
unfolding *conjoin-def same-length-def*

by *auto*
then have *clist-hd-tl*:*snd* (*clist*!*i*) = *hd* (*snd* (*clist*!*i*)) # *tl* (*snd* (*clist* ! *i*))
by *auto*
also have *hd* (*snd* (*clist*!*i*)) = (*snd* (*clist*!*i*))!*0*
using *a4′ lengz* **by** (*simp add*: *hd-conv-nth*)
ultimately have *clist-i-tl*:*snd* (*clist*!*i*) = (*xs*!*i*,*s*) # *tl* (*snd* (*clist* ! *i*))
using *clist-i-cptn* **by** *fastforce*
also have *tl* (*snd* (*clist* ! *i*)) = *map* (λ*x*. *tl* (*snd x*)) *clist*!*i*
using *nth-map a4′*
by *auto*
ultimately have *snd-clist*:*snd* (*clist*!*i*) = (*xs* ! *i*, *s*) # *map* (λ*x*. *tl* (*snd x*)) *clist* ! *i*
by *auto*
also have (*clist*!*i*) = (*fst* (*clist*!*i*),*snd* (*clist*!*i*))
by *auto*
ultimately have (*clist*!*i*) =(Γ, (*xs* ! *i*, *s*) # *map* (λ*x*. *tl* (*snd x*)) *clist* ! *i*)
using *clist-i-cptn* **by** *auto*
then have (Γ, (*xs* ! *i*, *s*) # *map* (λ*x*. *tl* (*snd x*)) *clist* ! *i*) ∈ *cptn*
using *clist-i-cptn* **by** *auto*
}
then have *clist-in-cptn*:(∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # ((*map* (λ*x*. *tl* (*snd x*))) *clist*) ! *i*) ∈ *cptn*)
by *auto*
have *same-length-clist-xs*:*length* ((*map* (λ*x*. *tl* (*snd x*))) *clist*) = *length xs*
using *props* **by** *auto*
then have (∃ *clist*. *length clist* = *length xs* ∧
(Γ, (*xs*, *s*) # *ys*) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*)) (*zip xs clist*) ∧
(∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*))
using *a1 props x-pair a-pair Cons a1-xs a2-s conjoin-clist-xs clist-in-cptn conjoin-map clist-map* **by** *blast*
then have (Γ, *c*) ∈ *par-cptn* **using** *a1 a2 props x-pair a-pair Cons a1-xs a2-s*
unfolding *par-cp-def* **by** *simp*
thus *x*∈*par-cp* Γ *xs s*
using *a1 a2 props x-pair a-pair Cons a1-xs a2-s*
unfolding *par-cp-def conjoin-def same-length-def same-program-def same-state-def same-functions-def compat-label-def*
by *simp*
qed
}
thus *?thesis* **using** *a1 a2* **by** *auto*
qed
}
qed

lemma *one-iff-aux-only-if*:$xs \neq []$ $\Longrightarrow$
(*par-cp* Γ (*xs*) *s* = {(Γ*1*,*c*). ∃ *clist*. (*length clist*)=(*length xs*) ∧
(∀ *i*<*length clist*. *clist*!*i* ∈ *cp* Γ (*xs*!*i*) *s*) ∧ (Γ,*c*) ∝ *clist* ∧ Γ*1*=Γ}) $\Longrightarrow$
(∀ *ys*. ((Γ,((*xs*, *s*)#*ys*)) ∈ *par-cptn*) =
(∃ *clist*. *length clist*= *length xs* ∧
((Γ,(*xs*, *s*)#*ys*) ∝ *map* (λ*i*. (Γ,(*fst i*,*s*)#(*snd i*))) (*zip xs clist*)) ∧
(∀ *i*<*length xs*. (Γ,(*xs*!*i*,*s*)#(*clist*!*i*)) ∈ *cptn*)))
proof
 fix *ys*
 assume *a1*: *xs*≠[]
 assume *a2*: *par-cp* Γ *xs s* =
   {(Γ*1*, *c*).
   ∃ *clist*.
     length clist = *length xs* ∧
     (∀ *i*<*length clist*.
       clist ! *i* ∈ *cp* Γ (*xs* ! *i*) *s*) ∧
     (Γ, *c*) ∝ *clist* ∧ Γ*1* = Γ}
 from *a1 a2* **show**
 ((Γ, (*xs*, *s*) # *ys*) ∈ *par-cptn*) =
   (∃ *clist*.
     length clist = *length xs* ∧
     (Γ,
     (*xs*, *s*) #
     ys) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
       (*zip xs clist*) ∧
     (∀ *i*<*length xs*.
       (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*))
 proof *auto*
  {assume *a3*:(Γ, (*xs*, *s*) # *ys*) ∈ *par-cptn*
  then show ∃ *clist*.
   length clist = *length xs* ∧
   (Γ,
   (*xs*, *s*) #
   ys) ∝ *map* (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
     (*zip xs clist*) ∧
   (∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*) ∈ *cptn*)
   using *a1 a2* **by** (*simp add*: *aux-onlyif*)
  }
  {fix *clist* ::(('*a*, '*b*, '*c*, '*d*) *LanguageCon.com* ×
    ('*a*, '*c*) *xstate*) *list list*
  assume *a3*: *length clist* = *length xs*
  assume *a4*:(Γ, (*xs*, *s*) # *ys*) ∝
     map (λ*i*. (Γ, (*fst i*, *s*) # *snd i*))
     (*zip xs clist*)
  assume *a5*: ∀ *i*<*length xs*. (Γ, (*xs* ! *i*, *s*) # *clist* ! *i*)
     ∈ *cptn*
  show (Γ, (*xs*, *s*) # *ys*) ∈ *par-cptn*
  using *a3 a4 a5* **using** *aux-if* **by** *blast*
  }

```
  qed
qed

lemma one-iff-aux: xs≠[] ⟹ (∀ ys. ((Γ,((xs, s)#ys)) ∈ par-cptn) =
 (∃ clist. length clist= length xs ∧
 ((Γ,(xs, s)#ys) ∝ map (λi. (Γ,(fst i,s)#(snd i))) (zip xs clist)) ∧
 (∀ i<length xs. (Γ,(xs!i,s)#(clist!i)) ∈ cptn))) =
 (par-cp Γ (xs) s = {(Γ1,c). ∃ clist. (length clist)=(length xs) ∧
 (∀ i<length clist. clist!i ∈ cp Γ (xs!i) s) ∧ (Γ,c) ∝ clist ∧ Γ1=Γ})
proof
  assume a1:xs≠[]
  {assume a2:(∀ ys. ((Γ,((xs, s)#ys)) ∈ par-cptn) =
   (∃ clist. length clist= length xs ∧
   ((Γ,(xs, s)#ys) ∝ map (λi. (Γ,(fst i,s)#(snd i))) (zip xs clist)) ∧
   (∀ i<length xs. (Γ,(xs!i,s)#(clist!i)) ∈ cptn)))
    then show (par-cp Γ (xs) s = {(Γ1,c). ∃ clist. (length clist)=(length xs) ∧
   (∀ i<length clist. clist!i ∈ cp Γ (xs!i) s) ∧ (Γ,c) ∝ clist ∧ Γ1=Γ})
    by (auto simp add: a1 a2 one-iff-aux-if)
  }
  {assume a2:(par-cp Γ (xs) s = {(Γ1,c). ∃ clist. (length clist)=(length xs) ∧
   (∀ i<length clist. clist!i ∈ cp Γ (xs!i) s) ∧ (Γ,c) ∝ clist ∧ Γ1=Γ})
    then show (∀ ys. ((Γ,((xs, s)#ys)) ∈ par-cptn) =
   (∃ clist. length clist= length xs ∧
   ((Γ,(xs, s)#ys) ∝ map (λi. (Γ,(fst i,s)#(snd i))) (zip xs clist)) ∧
   (∀ i<length xs. (Γ,(xs!i,s)#(clist!i)) ∈ cptn)))
   by (auto simp add: a1 a2 one-iff-aux-only-if)
  }
qed



theorem one:
xs≠[] ⟹
 par-cp Γ xs s =
    {(Γ1,c). ∃ clist. (length clist)=(length xs) ∧
            (∀ i<length clist. (clist!i) ∈ cp Γ (xs!i) s) ∧
            (Γ,c) ∝ clist ∧ Γ1=Γ}

apply(frule one-iff-aux)
apply(drule sym)
apply(erule iffD2)
apply clarify
apply(rule iffI)
 apply(erule aux-onlyif)
apply clarify
apply(force intro:aux-if)
done
```

end

9 Hoare Logic for Partial Correctness

theory *HoarePartialDef* **imports** *Semantic* **begin**

type-synonym $('s,'p)$ *quadruple* = $('s\ assn \times 'p \times 's\ assn \times 's\ assn)$

9.1 Validity of Hoare Tuples: $\Gamma,\Theta\models_{/F}\ P\ c\ Q,A$

definition
valid :: $[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] => bool$
$(\text{-}\models_{'/\text{-}}/\ \text{- - -,-}\ \ [61,60,1000,\ 20,\ 1000,1000]\ 60)$
where
$\Gamma\models_{/F}\ P\ c\ Q,A \equiv$
$\forall s\ t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \longrightarrow s \in Normal\ `\ P \longrightarrow$
$t \notin Fault\ `\ F \longrightarrow$
$t \in Normal\ `\ Q \cup Abrupt\ `\ A$

definition
cvalid::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] =>bool$
$(\text{-,-}\models_{'/\text{-}}/\ \text{- - -,-}\ \ [61,60,60,1000,\ 20,\ 1000,1000]\ 60)$
where
$\Gamma,\Theta\models_{/F}\ P\ c\ Q,A \equiv$
$(\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{/F}\ P\ (Call\ p)\ Q,A) \longrightarrow$
$\Gamma \models_{/F}\ P\ c\ Q,A$

definition
nvalid :: $[('s,'p,'f)\ body,nat,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] => bool$
$(\text{-}\models\text{-:}_{'/\text{-}}/\ \text{- - -,-}\ \ [61,60,60,1000,\ 20,\ 1000,1000]\ 60)$
where
$\Gamma\models n:_{/F}\ P\ c\ Q,A \equiv \forall s\ t.\ \Gamma\vdash\langle c,s\ \rangle =n\Rightarrow t \longrightarrow s \in Normal\ `\ P \longrightarrow t \notin Fault\ `$
F
$\longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A$

definition
cnvalid::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,nat,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] \Rightarrow bool$

(-,-⊨-:′/-/ - - -,- [61,60,60,60,1000, 20, 1000,1000] 60)

where

$\Gamma,\Theta\models n:_{/F}\ P\ c\ Q,A \equiv (\forall (P,p,Q,A)\in\Theta.\ \Gamma\models n:_{/F}\ P\ (Call\ p)\ Q,A) \longrightarrow \Gamma\ \models n:_{/F}\ P\ c\ Q,A$

notation (*ASCII*)
valid (-|=′/-/ - - -,- [61,60,1000, 20, 1000,1000] 60) **and**
cvalid (-,-|=′/-/ - - -,- [61,60,60,1000, 20, 1000,1000] 60) **and**
nvalid (-|=-:′/-/ - - -,- [61,60,60,1000, 20, 1000,1000] 60) **and**
cnvalid (-,-|=-:′/-/ - - -,- [61,60,60,60,1000, 20, 1000,1000] 60)

9.2 Properties of Validity

lemma *valid-iff-nvalid*: $\Gamma\models_{/F}\ P\ c\ Q,A = (\forall n.\ \Gamma\models n:_{/F}\ P\ c\ Q,A)$
apply (*simp only*: *valid-def nvalid-def exec-iff-execn*)
apply (*blast dest*: *exec-final-notin-to-execn*)
done

lemma *cnvalid-to-cvalid*: $(\forall n.\ \Gamma,\Theta\models n:_{/F}\ P\ c\ Q,A) \Longrightarrow \Gamma,\Theta\models_{/F}\ P\ c\ Q,A$
apply (*unfold cvalid-def cnvalid-def valid-iff-nvalid* [*THEN eq-reflection*])
apply *fast*
done

lemma *nvalidI*:
$[\![\bigwedge s\ t.\ [\![\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow\ t;s \in P;\ t\notin\ Fault\ `\ F]\!] \Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A]\!]$
$\Longrightarrow \Gamma\models n:_{/F}\ P\ c\ Q,A$
by (*auto simp add*: *nvalid-def*)

lemma *validI*:
$[\![\bigwedge s\ t.\ [\![\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow\ t;s \in P;\ t\notin Fault\ `\ F]\!] \Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A]\!]$
$\Longrightarrow \Gamma\models_{/F}\ P\ c\ Q,A$
by (*auto simp add*: *valid-def*)

lemma *cvalidI*:
$[\![\bigwedge s\ t.\ [\![\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{/F}\ P\ (Call\ p)\ Q,A;\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t;s \in P;t\notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A]\!]$
$\Longrightarrow \Gamma,\Theta\models_{/F}\ P\ c\ Q,A$
by (*auto simp add*: *cvalid-def valid-def*)

lemma *cvalidD*:
$[\![\Gamma,\Theta\models_{/F}\ P\ c\ Q,A;\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{/F}\ P\ (Call\ p)\ Q,A;\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t;s \in P;t\notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A$
by (*auto simp add*: *cvalid-def valid-def*)

lemma *cnvalidI*:
$\llbracket\bigwedge s\ t.\ \llbracket\forall (P,p,Q,A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A;$
$\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow t;s\in P;t\notin Fault\ `\ F\rrbracket$
$\Longrightarrow t\in Normal\ `\ Q\cup Abrupt\ `\ A\rrbracket$
$\Longrightarrow \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
by (*auto simp add*: *cnvalid-def nvalid-def*)

lemma *cnvalidD*:
$\llbracket\Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A;\forall (P,p,Q,A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A;$
$\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow t;s\in P;$
$t\notin Fault\ `\ F\rrbracket$
$\Longrightarrow t\in Normal\ `\ Q\cup Abrupt\ `\ A$
by (*auto simp add*: *cnvalid-def nvalid-def*)

lemma *nvalid-augment-Faults*:
assumes *validn*:$\Gamma\models n{:}_{/F}\ P\ c\ Q,A$
assumes F': $F\subseteq F'$
shows $\Gamma\models n{:}_{/F'}\ P\ c\ Q,A$
proof (*rule nvalidI*)
fix *s t*
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow t$
assume P: $s\in P$
assume F: $t\notin Fault\ `\ F'$
with F' **have** $t\notin Fault\ `\ F$
by *blast*
with *exec P validn*
show $t\in Normal\ `\ Q\cup Abrupt\ `\ A$
by (*auto simp add*: *nvalid-def*)
qed

lemma *valid-augment-Faults*:
assumes *validn*:$\Gamma\models_{/F}\ P\ c\ Q,A$
assumes F': $F\subseteq F'$
shows $\Gamma\models_{/F'}\ P\ c\ Q,A$
proof (*rule validI*)
fix *s t*
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow t$
assume P: $s\in P$
assume F: $t\notin Fault\ `\ F'$
with F' **have** $t\notin Fault\ `\ F$
by *blast*
with *exec P validn*
show $t\in Normal\ `\ Q\cup Abrupt\ `\ A$
by (*auto simp add*: *valid-def*)
qed

lemma *nvalid-to-nvalid-strip*:
  assumes *validn*:$\Gamma\models n{:}_{/F}$ *P c Q*,*A*
  assumes F': $F' \subseteq -F$
  shows *strip* F' $\Gamma\models n{:}_{/F}$ *P c Q*,*A*
proof (*rule nvalidI*)
  fix *s t*
  assume *exec-strip*: *strip* F' $\Gamma\vdash\langle c,\mathit{Normal}\ s\ \rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *F*: $t \notin \mathit{Fault}$ ‘ *F*
  from *exec-strip* **obtain** t' **where**
    exec: $\Gamma\vdash\langle c,\mathit{Normal}\ s\ \rangle =n\Rightarrow t'$ **and**
    t': $t' \in \mathit{Fault}$ ‘ $(-F') \longrightarrow t'=t\ \neg\ \mathit{isFault}\ t' \longrightarrow t'=t$
    by (*blast dest*: *execn-strip-to-execn*)
  show $t \in \mathit{Normal}$ ‘ *Q* $\cup$ *Abrupt* ‘ *A*
  proof (*cases* $t' \in \mathit{Fault}$ ‘ *F*)
    case *True*
    with t' *F* F' **have** *False*
      by *blast*
    thus *?thesis* **..**
  next
    case *False*
    with *exec P validn*
    have $t' \in \mathit{Normal}$ ‘ *Q* $\cup$ *Abrupt* ‘ *A*
      by (*auto simp add*: *nvalid-def*)
    moreover
    from *this* t' **have** $t'=t$
      by *auto*
    ultimately show *?thesis*
      by *simp*
  qed
qed

lemma *valid-to-valid-strip*:
  assumes *valid*:$\Gamma\models_{/F}$ *P c Q*,*A*
  assumes F': $F' \subseteq -F$
  shows *strip* F' $\Gamma\models_{/F}$ *P c Q*,*A*
proof (*rule validI*)
  fix *s t*
  assume *exec-strip*: *strip* F' $\Gamma\vdash\langle c,\mathit{Normal}\ s\ \rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *F*: $t \notin \mathit{Fault}$ ‘ *F*
  from *exec-strip* **obtain** t' **where**
    exec: $\Gamma\vdash\langle c,\mathit{Normal}\ s\ \rangle \Rightarrow t'$ **and**
    t': $t' \in \mathit{Fault}$ ‘ $(-F') \longrightarrow t'=t\ \neg\ \mathit{isFault}\ t' \longrightarrow t'=t$
    by (*blast dest*: *exec-strip-to-exec*)
  show $t \in \mathit{Normal}$ ‘ *Q* $\cup$ *Abrupt* ‘ *A*
  proof (*cases* $t' \in \mathit{Fault}$ ‘ *F*)
    case *True*

with t' F F' **have** *False*
by *blast*
thus *?thesis* **..**
next
case *False*
with *exec P valid*
have $t' \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
by (*auto simp add*: *valid-def*)
moreover
from *this* t' **have** $t'{=}t$
by *auto*
ultimately show *?thesis*
by *simp*
qed
qed

9.3 The Hoare Rules: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$

lemma *mono-WeakenContext*: $A \subseteq B \Longrightarrow$
$(\lambda(P,\ c,\ Q,\ A').\ (\Gamma,\ \Theta,\ F,\ P,\ c,\ Q,\ A') \in A)\ x \longrightarrow$
$(\lambda(P,\ c,\ Q,\ A').\ (\Gamma,\ \Theta,\ F,\ P,\ c,\ Q,\ A') \in B)\ x$
apply *blast*
done

inductive *hoarep*::[('s,'p,'f) body,('s,'p) quadruple set,'f set,
's assn,('s,'p,'f) com, 's assn,'s assn] => bool
((3-,-/⊢'/- (-/ (-)/ -,/-)) [60,60,60,1000,20,1000,1000]60)
for Γ::('s,'p,'f) *body*
where
Skip: $\Gamma,\Theta\vdash_{/F}\ Q\ \mathit{Skip}\ Q,A$

| *Basic*: $\Gamma,\Theta\vdash_{/F}\ \{s.\ f\ s \in Q\}\ (\mathit{Basic}\ f)\ Q,A$

| *Spec*: $\Gamma,\Theta\vdash_{/F}\ \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}\ (\mathit{Spec}\ r)\ Q,A$

| *Seq*: $[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Seq}\ c_1\ c_2)\ Q,A$

| *Cond*: $[\![\Gamma,\Theta\vdash_{/F}\ (P \cap b)\ c_1\ Q,A;\ \Gamma,\Theta\vdash_{/F}\ (P \cap -\ b)\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Cond}\ b\ c_1\ c_2)\ Q,A$

| *While*: $\Gamma,\Theta\vdash_{/F}\ (P \cap b)\ c\ P,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{While}\ b\ c)\ (P \cap -\ b),A$

| *Guard*: $\Gamma,\Theta\vdash_{/F}\ (g \cap P)\ c\ Q,A$

$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ $(g \cap P)$ $(Guard\ f\ g\ c)$ Q,A

| *Guarantee*: $[\![f \in F;\ \Gamma,\Theta\vdash_{/F}\ (g \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (Guard\ f\ g\ c)\ Q,A$

| *CallRec*:
$[\![(P,p,Q,A) \in Specs;$
$\forall (P,p,Q,A) \in Specs.\ p \in dom\ \Gamma \wedge \Gamma,\Theta \cup Specs\vdash_{/F}\ P\ (the\ (\Gamma\ p))\ Q,A\ ]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Call\ p)\ Q,A$

| *DynCom*:
$\forall s \in P.\ \Gamma,\Theta\vdash_{/F}\ P\ (c\ s)\ Q,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (DynCom\ c)\ Q,A$

| *Throw*: $\Gamma,\Theta\vdash_{/F}\ A\ Throw\ Q,A$

| *Catch*: $[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ Q,R;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!] \Longrightarrow\ \Gamma,\Theta\vdash_{/F}\ P\ Catch\ c_1\ c_2\ Q,A$

| *Conseq*: $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F}\ P'\ c\ Q',A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$

| *Asm*: $[\![(P,p,Q,A) \in \Theta]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (Call\ p)\ Q,A$

| *ExFalso*: $[\![\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A;\ \neg\ \Gamma\models_{/F}\ P\ c\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
— This is a hack rule that enables us to derive completeness for an arbitrary context Θ, from completeness for an empty context.

Does not work, because of rule ExFalso, the context Θ is to blame. A weaker version with empty context can be derived from soundness and completeness later on.

lemma *hoare-strip-*Γ:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ p\ Q,A$
shows *strip* $(-F)$ $\Gamma,\Theta\vdash_{/F}\ P\ p\ Q,A$
using *deriv*
proof *induct*
case *Skip* **thus** *?case* **by** (*iprover intro*: *hoarep.Skip*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *hoarep.Basic*)
next
case *Spec* **thus** *?case* **by** (*iprover intro*: *hoarep.Spec*)
next

case *Seq* **thus** *?case* **by** (*iprover intro*: *hoarep.Seq*)
next
case *Cond* **thus** *?case* **by** (*iprover intro*: *hoarep.Cond*)
next
case *While* **thus** *?case* **by** (*iprover intro*: *hoarep.While*)
next
case *Guard* **thus** *?case* **by** (*iprover intro*: *hoarep.Guard*)

next
case *DynCom*
thus *?case*
by − (*rule hoarep.DynCom*,*best elim*!: *ballE exE*)
next
case *Throw* **thus** *?case* **by** (*iprover intro*: *hoarep.Throw*)
next
case *Catch* **thus** *?case* **by** (*iprover intro*: *hoarep.Catch*)

next
case *Asm* **thus** *?case* **by** (*iprover intro*: *hoarep.Asm*)
next
case *ExFalso*
thus *?case*
oops

lemma *hoare-augment-context*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F} P\ p\ Q,A$
shows $\bigwedge\Theta'.\ \Theta \subseteq \Theta' \Longrightarrow \Gamma,\Theta'\vdash_{/F} P\ p\ Q,A$
using *deriv*
proof (*induct*)
case *CallRec*
case (*CallRec* $P\ p\ Q\ A$ *Specs* $\Theta\ F\ \Theta'$)
from *CallRec.prems*
have $\Theta\cup$*Specs*
$\subseteq \Theta'\cup$*Specs*
by *blast*
with *CallRec.hyps* (*2*)
have $\forall (P,p,Q,A)\in$*Specs*. $p \in dom\ \Gamma \wedge \Gamma,\Theta'\cup Specs \vdash_{/F} P\ \ (the\ (\Gamma\ p))\ Q,A$
by *fastforce*

with *CallRec* **show** *?case* **by** − (*rule hoarep.CallRec*)
next
case *DynCom* **thus** *?case* **by** (*blast intro*: *hoarep.DynCom*)
next
case (*Conseq* $P\ \Theta\ F\ c\ Q\ A\ \Theta'$)
from *Conseq*
have $\forall s \in P.$
$(\exists P'\ Q'\ A'.\ \Gamma,\Theta' \vdash_{/F} P'\ c\ Q',A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
with *Conseq* **show** *?case* **by** − (*rule hoarep.Conseq*)

next
  case $(ExFalso\ \Theta\ F\ P\ c\ Q\ A\ \Theta')$
  have *valid-ctxt*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c\ Q,A\ \Theta \subseteq \Theta'$ **by** *fact+*
  hence $\forall n.\ \Gamma,\Theta' \models n{:}_{/F}\ P\ c\ Q,A$
    by (*simp add*: *cnvalid-def*) *blast*
  moreover have *invalid*: $\neg\ \Gamma \models_{/F}\ P\ c\ Q,A$ **by** *fact*
  ultimately show *?case*
    by (*rule hoarep.ExFalso*)
qed (*blast intro*: *hoarep.intros*)+

9.4 Some Derived Rules

lemma *Conseq'*: $\forall s.\ s \in P \longrightarrow$
$(\exists P'\ Q'\ A'.$
$(\forall\ Z.\ \Gamma,\Theta \vdash_{/F}\ (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)) \wedge$
$(\exists Z.\ s \in P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A)))$
$\Longrightarrow$
$\Gamma,\Theta \vdash_{/F}\ P\ c\ Q,A$
apply (*rule Conseq*)
apply (*rule ballI*)
apply (*erule-tac* $x{=}s$ **in** *allE*)
apply (*clarify*)
apply (*rule-tac* $x{=}P'\ Z$ **in** *exI*)
apply (*rule-tac* $x{=}Q'\ Z$ **in** *exI*)
apply (*rule-tac* $x{=}A'\ Z$ **in** *exI*)
apply *blast*
done

lemma *conseq*: $[\![\forall Z.\ \Gamma,\Theta \vdash_{/F}\ (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\forall s.\ s \in P \longrightarrow (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\Longrightarrow$
$\Gamma,\Theta \vdash_{/F}\ P\ c\ Q,A$
  by (*rule Conseq*) *blast*

theorem *conseqPrePost* [*trans*]:
  $\Gamma,\Theta \vdash_{/F}\ P'\ c\ Q',A' \Longrightarrow P \subseteq P' \Longrightarrow\ Q' \subseteq Q \Longrightarrow A' \subseteq A \Longrightarrow\ \Gamma,\Theta \vdash_{/F}\ P\ c\ Q,A$
  by (*rule conseq* [**where** *?P'*$=\lambda Z.\ P'$ **and** *?Q'*$=\lambda Z.\ Q'$]) *auto*

lemma *conseqPre* [*trans*]: $\Gamma,\Theta \vdash_{/F}\ P'\ c\ Q,A \Longrightarrow P \subseteq P' \Longrightarrow \Gamma,\Theta \vdash_{/F}\ P\ c\ Q,A$
by (*rule conseq*) *auto*

lemma *conseqPost* [*trans*]: $\Gamma,\Theta \vdash_{/F}\ P\ c\ Q',A' \Longrightarrow Q' \subseteq Q \Longrightarrow A' \subseteq A$
$\Longrightarrow\ \ \Gamma,\Theta \vdash_{/F}\ P\ c\ Q,A$
  by (*rule conseq*) *auto*

lemma *CallRec'*:
  $[\![p{\in}Procs;\ Procs \subseteq dom\ \Gamma;$

$\forall\, p{\in}Procs.$
$\forall\, Z.\ \Gamma,\Theta \cup (\bigcup p{\in}Procs.\ \bigcup Z.\ \{((P\ p\ Z),p,Q\ p\ Z,A\ p\ Z)\})$
$\vdash_{/F}\ (P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ (P\ p\ Z)\ (Call\ p)\ (Q\ p\ Z),(A\ p\ Z)$
apply (*rule CallRec* [**where** *Specs*=$\bigcup p{\in}Procs.\ \bigcup Z.\ \{((P\ p\ Z),p,Q\ p\ Z,A\ p\ Z)\}$])
apply *blast*
apply *blast*
done

end

10 Properties of Partial Correctness Hoare Logic

theory *HoarePartialProps* **imports** *HoarePartialDef* **begin**

10.1 Soundness

lemma *hoare-cnvalid*:
 assumes *hoare*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
 shows $\bigwedge n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
using *hoare*
proof (*induct*)
  case (*Skip* Θ *F P A*)
  show $\Gamma,\Theta \models n{:}_{/F}\ P\ Skip\ P,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume $\Gamma\vdash\langle Skip,Normal\ s\rangle = n\Rightarrow\ t\ s \in P$
    thus $t \in Normal\ \text{`}\ P \cup Abrupt\ \text{`}\ A$
      by *cases auto*
  qed
next
  case (*Basic* Θ *F f P A*)
  show $\Gamma,\Theta \models n{:}_{/F}\ \{s.\ f\ s \in P\}\ (Basic\ f)\ P,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume $\Gamma\vdash\langle Basic\ f,Normal\ s\rangle = n\Rightarrow\ t\ s \in \{s.\ f\ s \in P\}$
    thus $t \in Normal\ \text{`}\ P \cup Abrupt\ \text{`}\ A$
      by *cases auto*
  qed
next
  case (*Spec* Θ *F r Q A*)
  show $\Gamma,\Theta\models n{:}_{/F}\ \{s.\ (\forall\, t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists\, t.\ (s,\ t) \in r)\}\ Spec\ r\ Q,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume *exec*: $\Gamma\vdash\langle Spec\ r,Normal\ s\rangle = n\Rightarrow\ t$
    assume *P*: $s \in \{s.\ (\forall\, t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists\, t.\ (s,\ t) \in r)\}$
    from *exec P*

show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
by *cases auto*
qed
next
case (*Seq* Θ *F P c1 R A c2 Q*)
have *valid-c1*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c1\ R,A$ **by** *fact*
have *valid-c2*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ R\ c2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ P$ *Seq c1 c2 Q*,A
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle =n\Rightarrow t$
assume *t-notin-F*: $t \notin$ *Fault* ‘ F
assume *P*: $s \in P$
from *exec P* **obtain** *r* **where**
exec-c1: $\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow r$ **and** *exec-c2*: $\Gamma\vdash\langle c2,r\rangle =n\Rightarrow t$
by *cases auto*
with *t-notin-F* **have** $r \notin$ *Fault* ‘ F
by (*auto dest*: *execn-Fault-end*)
with *valid-c1 ctxt exec-c1 P*
have *r*: $r{\in}$*Normal* ‘ $R \cup$ *Abrupt* ‘ A
by (*rule cnvalidD*)
show $t{\in}$*Normal* ‘ $Q \cup$ *Abrupt* ‘ A
proof (*cases r*)
case (*Normal* r')
with *exec-c2 r*
show $t{\in}$*Normal* ‘ $Q \cup$ *Abrupt* ‘ A
apply $-$
apply (*rule cnvalidD* [*OF valid-c2 ctxt - - t-notin-F*])
apply *auto*
done
next
case (*Abrupt* r')
with *exec-c2* **have** t=*Abrupt* r'
by (*auto elim*: *execn-elim-cases*)
with *Abrupt r* **show** *?thesis*
by *auto*
next
case *Fault* **with** *r* **show** *?thesis* **by** *blast*
next
case *Stuck* **with** *r* **show** *?thesis* **by** *blast*
qed
qed
next
case (*Cond* Θ *F P b c1 Q A c2*)
have *valid-c1*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ (P \cap b)\ c1\ Q,A$ **by** *fact*
have *valid-c2*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ (P \cap -\ b)\ c2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ P$ *Cond b c1 c2 Q*,A

proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q{,}A$
assume *exec*: $\Gamma \vdash \langle Cond\ b\ c1\ c2{,}Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault\ `\ F$
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
proof (*cases* $s \in b$)
case *True*
with *exec* **have** $\Gamma \vdash \langle c1{,}Normal\ s\rangle =n\Rightarrow t$
by *cases auto*
with *P True*
show *?thesis*
by − (*rule cnvalidD* [*OF valid-c1 ctxt - - t-notin-F*],*auto*)
next
case *False*
with *exec P* **have** $\Gamma \vdash \langle c2{,}Normal\ s\rangle =n\Rightarrow t$
by *cases auto*
with *P False*
show *?thesis*
by − (*rule cnvalidD* [*OF valid-c2 ctxt - - t-notin-F*],*auto*)
qed
qed
next
case (*While* Θ *F P b c A n*)
have *valid-c*: $\bigwedge n.\ \Gamma{,}\Theta \models n{:}_{/F}\ (P \cap b)\ c\ P{,}A$ **by** *fact*
show $\Gamma{,}\Theta \models n{:}_{/F}\ P\ While\ b\ c\ (P \cap -\ b){,}A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q{,}A$
assume *exec*: $\Gamma \vdash \langle While\ b\ c{,}Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault\ `\ F$
show $t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$
proof (*cases* $s \in b$)
case *True*
{
fix $d{::}('b,'a,'c)$ *com* **fix** *s t*
assume *exec*: $\Gamma \vdash \langle d{,}s\rangle =n\Rightarrow t$
assume *d*: $d = While\ b\ c$
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q{,}A$
from *exec d ctxt*
have $[\![s \in Normal\ `\ P;\ t \notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ (P \cap -\ b) \cup Abrupt`A$
proof (*induct*)
case (*WhileTrue s* b' c' *n r t*)
have *t-notin-F*: $t \notin Fault\ `\ F$ **by** *fact*
have *eqs*: $While\ b'\ c' = While\ b\ c$ **by** *fact*

note *valid-c*
moreover have *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$ **by** *fact*
moreover from *WhileTrue*
obtain $\Gamma \vdash \langle c, Normal\ s\rangle =n\Rightarrow r$ **and**
$\Gamma \vdash \langle While\ b\ c, r\rangle =n\Rightarrow t$ **and**
$Normal\ s \in Normal\ `(P \cap b)$ **by** *auto*
moreover with *t-notin-F* **have** $r \notin Fault\ `\ F$
by (*auto dest*: *execn-Fault-end*)
ultimately
have *r*: $r \in Normal\ `\ P \cup Abrupt\ `\ A$
by − (*rule cnvalidD*,*auto*)
from *this* - *ctxt*
show $t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$
proof (*cases r*)
case (*Normal* r')
with *r ctxt eqs t-notin-F*
show *?thesis*
by − (*rule WhileTrue.hyps*,*auto*)
next
case (*Abrupt* r')
have $\Gamma \vdash \langle While\ b'\ c', r\rangle =n\Rightarrow t$ **by** *fact*
with *Abrupt* **have** $t=r$
by (*auto dest*: *execn-Abrupt-end*)
with *r Abrupt* **show** *?thesis*
by *blast*
next
case *Fault* **with** *r* **show** *?thesis* **by** *blast*
next
case *Stuck* **with** *r* **show** *?thesis* **by** *blast*
qed
qed *auto*
}
with *exec ctxt P t-notin-F*
show *?thesis*
by *auto*
next
case *False*
with *exec P* **have** $t=Normal\ s$
by *cases auto*
with *P False*
show *?thesis*
by *auto*
qed
qed
next
case (*Guard* Θ *F g P c Q A f*)
have *valid-c*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ (g \cap P)\ c\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ (g \cap P)\ Guard\ f\ g\ c\ \ Q,A$
proof (*rule cnvalidI*)

fix s t
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma \vdash \langle Guard\ f\ g\ c, Normal\ s\rangle =n\Rightarrow t$
assume *t-notin-F*: $t \notin Fault\ `\ F$
assume $P{:}s \in (g \cap P)$
from *exec P* **have** $\Gamma \vdash \langle c, Normal\ s\rangle =n\Rightarrow t$
by *cases auto*
from *valid-c ctxt this P t-notin-F*
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
by (*rule cnvalidD*)
qed
next
case (*Guarantee f F* Θ *g P c Q A*)
have *valid-c*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ (g \cap P)\ c\ Q,A$ **by** *fact*
have *f-F*: $f \in F$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ P\ Guard\ f\ g\ c\ \ Q,A$
proof (*rule cnvalidI*)
fix s t
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma \vdash \langle Guard\ f\ g\ c, Normal\ s\rangle =n\Rightarrow t$
assume *t-notin-F*: $t \notin Fault\ `\ F$
assume $P{:}s \in P$
from *exec f-F t-notin-F* **have** g: $s \in g$
by *cases auto*
with P **have** P': $s \in g \cap P$
by *blast*
from *exec P g* **have** $\Gamma \vdash \langle c, Normal\ s\rangle =n\Rightarrow t$
by *cases auto*
from *valid-c ctxt this P' t-notin-F*
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
by (*rule cnvalidD*)
qed
next
case (*CallRec P p Q A Specs* Θ *F*)
have p: $(P,p,Q,A) \in Specs$ **by** *fact*
have *valid-body*:
$\forall (P,p,Q,A) \in Specs.\ p \in dom\ \Gamma \wedge (\forall n.\ \Gamma,\Theta \cup Specs \models n{:}_{/F}\ P\ (the\ (\Gamma\ p))\ Q,A)$
using *CallRec.hyps* **by** *blast*
show $\Gamma,\Theta \models n{:}_{/F}\ P\ Call\ p\ Q,A$
proof −
{
fix n
have $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
$\Longrightarrow \forall (P,p,Q,A) \in Specs.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
proof (*induct n*)
case *0*
show $\forall (P,p,Q,A) \in Specs.\ \Gamma \models 0{:}_{/F}\ P\ (Call\ p)\ Q,A$

```
      by (fastforce elim!: execn-elim-cases simp add: nvalid-def)
  next
    case (Suc m)
    have hyp: ∀(P, p, Q, A)∈Θ. Γ ⊨m:/F P (Call p) Q,A
          ⟹ ∀(P,p,Q,A) ∈Specs. Γ⊨m:/F P (Call p) Q,A by fact
    have ∀(P, p, Q, A)∈Θ. Γ ⊨Suc m:/F P (Call p) Q,A by fact
    hence ctxt-m: ∀(P, p, Q, A)∈Θ. Γ ⊨m:/F P (Call p) Q,A
      by (fastforce simp add: nvalid-def intro: execn-Suc)
    hence valid-Proc:
      ∀(P,p,Q,A) ∈Specs. Γ⊨m:/F P (Call p) Q,A
      by (rule hyp)
    let ?Θ'= Θ ∪ Specs
    from valid-Proc ctxt-m
    have ∀(P, p, Q, A)∈?Θ'. Γ ⊨m:/F P (Call p) Q,A
      by fastforce
    with valid-body
    have valid-body-m:
      ∀(P,p,Q,A) ∈Specs. ∀n. Γ ⊨m:/F P (the (Γ p)) Q,A
      by (fastforce simp add: cnvalid-def)
    show ∀(P,p,Q,A) ∈Specs. Γ ⊨Suc m:/F P (Call p) Q,A
    proof (clarify)
      fix P p Q A assume p: (P,p,Q,A) ∈ Specs
      show Γ ⊨Suc m:/F P (Call p) Q,A
      proof (rule nvalidI)
        fix s t
        assume exec-call:
          Γ⊢⟨Call p,Normal s⟩ =Suc m⇒ t
        assume Pre: s ∈ P
        assume t-notin-F: t ∉ Fault ‘ F
        from exec-call
        show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
        proof (cases)
          fix bdy m'
          assume m: Suc m = Suc m'
          assume bdy: Γ p = Some bdy
          assume exec-body: Γ⊢⟨bdy,Normal s⟩ =m'⇒ t
          from Pre valid-body-m exec-body bdy m p t-notin-F
          show ?thesis
            by (fastforce simp add: nvalid-def)
        next
          assume Γ p = None
          with valid-body p have False by auto
          thus ?thesis ..
        qed
      qed
    qed
  qed
}
```

with *p* **show** *?thesis*
by (*fastforce simp add*: *cnvalid-def*)
qed
next
case (*DynCom P* Θ *F c Q A*)
hence *valid-c*: $\forall s{\in}P.\ (\forall n.\ \Gamma,\Theta{\models}n{:}_{/F}\ P\ (c\ s)\ Q,A)$ **by** *auto*
show $\Gamma,\Theta{\models}n{:}_{/F}\ P\ DynCom\ c\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\ {\models}n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma{\vdash}\langle DynCom\ c,Normal\ s\rangle\ ={n}{\Rightarrow}\ t$
assume *P*: $s \in P$
assume *t-notin-Fault*: $t \notin Fault\ \text{`}\ F$
from *exec* **show** $t \in Normal\ \text{`}\ Q \cup Abrupt\ \text{`}\ A$
proof (*cases*)
assume $\Gamma{\vdash}\langle c\ s,Normal\ s\rangle\ ={n}{\Rightarrow}\ t$
from *cnvalidD* [*OF valid-c* [*rule-format*, *OF P*] *ctxt this P t-notin-Fault*]
show *?thesis* **.**
qed
qed
next
case (*Throw* Θ *F A Q*)
show $\Gamma,\Theta\ {\models}n{:}_{/F}\ A\ Throw\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume $\Gamma{\vdash}\langle Throw,Normal\ s\rangle\ ={n}{\Rightarrow}\ t\ s \in A$
then show $t \in Normal\ \text{`}\ Q \cup Abrupt\ \text{`}\ A$
by *cases simp*
qed
next
case (*Catch* Θ *F P* c_1 *Q R* c_2 *A*)
have *valid-c1*: $\bigwedge n.\ \Gamma,\Theta\ {\models}n{:}_{/F}\ P\ c_1\ Q,R$ **by** *fact*
have *valid-c2*: $\bigwedge n.\ \Gamma,\Theta\ {\models}n{:}_{/F}\ R\ c_2\ Q,A$ **by** *fact*
show $\Gamma,\Theta\ {\models}n{:}_{/F}\ P\ Catch\ c_1\ c_2\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\ {\models}n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma{\vdash}\langle Catch\ c_1\ c_2,Normal\ s\rangle\ ={n}{\Rightarrow}\ t$
assume *P*: $s \in P$
assume *t-notin-Fault*: $t \notin Fault\ \text{`}\ F$
from *exec* **show** $t \in Normal\ \text{`}\ Q \cup Abrupt\ \text{`}\ A$
proof (*cases*)
fix s'
assume *exec-c1*: $\Gamma{\vdash}\langle c_1,Normal\ s\rangle\ ={n}{\Rightarrow}\ Abrupt\ s'$
assume *exec-c2*: $\Gamma{\vdash}\langle c_2,Normal\ s'\rangle\ ={n}{\Rightarrow}\ t$
from *cnvalidD* [*OF valid-c1 ctxt exec-c1 P*]
have $Abrupt\ s' \in Abrupt\ \text{`}\ R$
by *auto*

```
      with cnvalidD [OF valid-c2 ctxt - - t-notin-Fault] exec-c2
      show ?thesis
        by fastforce
    next
      assume exec-c1: Γ⊢⟨c₁,Normal s⟩ =n⇒ t
      assume notAbr: ¬ isAbr t
      from cnvalidD [OF valid-c1 ctxt exec-c1 P t-notin-Fault]
      have t ∈ Normal ‘ Q ∪ Abrupt ‘ R .
      with notAbr
      show ?thesis
        by auto
    qed
  qed
next
  case (Conseq P Θ F c Q A)
  hence adapt: ∀ s ∈ P. (∃ P' Q' A'. Γ,Θ ⊨n:/F P' c Q',A'  ∧
                        s ∈ P' ∧ Q' ⊆ Q ∧ A' ⊆ A)
    by blast
  show Γ,Θ ⊨n:/F P c Q,A
  proof (rule cnvalidI)
    fix s t
    assume ctxt:∀ (P, p, Q, A)∈Θ. Γ⊨n:/F P (Call p) Q,A
    assume exec: Γ⊢⟨c,Normal s⟩ =n⇒ t
    assume P: s ∈ P
    assume t-notin-F: t ∉ Fault ‘ F
    show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
    proof −
      from P adapt obtain P' Q' A' Z  where
        spec: Γ,Θ⊨n:/F P' c Q',A' and
        P': s ∈ P'  and  strengthen: Q' ⊆ Q ∧ A' ⊆ A
        by auto
      from spec [rule-format] ctxt exec P' t-notin-F
      have t ∈ Normal ‘ Q' ∪ Abrupt ‘ A'
        by (rule cnvalidD)
      with strengthen show ?thesis
        by blast
    qed
  qed
next
  case (Asm P p Q A Θ F)
  have asm: (P, p, Q, A) ∈ Θ by fact
  show Γ,Θ ⊨n:/F P (Call p) Q,A
  proof (rule cnvalidI)
    fix s t
    assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ ⊨n:/F P (Call p) Q,A
    assume exec: Γ⊢⟨Call p,Normal s⟩ =n⇒ t
    from asm ctxt have Γ ⊨n:/F P Call p Q,A by auto
    moreover
```

assume $s \in P$ $t \notin$ *Fault* ‘ F
ultimately
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
using *exec*
by (*auto simp add*: *nvalid-def*)
qed
next
case *ExFalso* **thus** *?case* **by** *iprover*
qed

theorem *hoare-sound*: $\Gamma,\Theta\vdash_{/F}$ P c $Q,A \Longrightarrow \Gamma,\Theta\models_{/F}$ P c Q,A
by (*iprover intro*: *cnvalid-to-cvalid hoare-cnvalid*)

10.2 Completeness

lemma *MGT-valid*:
$\Gamma\models_{/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup\ \mathit{Fault}\ `\ (-F))\}\ c$
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}, \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule validI*)
fix s t
assume $\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow t$
$s \in \{s.\ s = Z \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup\ \mathit{Fault}\ `\ (-F))\}$
$t \notin \mathit{Fault}\ `\ F$
thus $t \in \mathit{Normal}\ `\ \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\} \cup$
$\mathit{Abrupt}\ `\ \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*cases* t) (*auto simp add*: *final-notin-def*)
qed

The consequence rule where the existential Z is instantiated to s. Usefull in proof of *MGT-lemma*.

lemma *ConseqMGT*:
assumes *modif*: $\forall Z.\ \Gamma,\Theta \vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
assumes *impl*: $\bigwedge s.\ s \in P \Longrightarrow s \in P'\ s \wedge (\forall t.\ t \in Q'\ s \longrightarrow t \in Q) \wedge$
$(\forall t.\ t \in A'\ s \longrightarrow t \in A)$
shows $\Gamma,\Theta \vdash_{/F} P\ c\ Q,A$
using *impl*
by − (*rule conseq* [*OF modif*],*blast*)

lemma *Seq-NoFaultStuckD1*:
assumes *noabort*: $\Gamma\vdash\langle \mathit{Seq}\ c1\ c2,s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F)$
shows $\Gamma\vdash\langle c1,s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F)$
proof (*rule final-notinI*)
fix t
assume *exec-c1*: $\Gamma\vdash\langle c1,s\rangle \Rightarrow t$
show $t \notin \{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F$
proof
assume $t \in \{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ F$
moreover

```
    {
      assume t = Stuck
      with exec-c1
      have Γ⊢⟨Seq c1 c2,s⟩ ⇒ Stuck
        by (auto intro: exec-Seq')
      with noabort have False
        by (auto simp add: final-notin-def)
      hence False ..
    }
    moreover
    {
      assume t ∈ Fault ` F
      then obtain f where
      t: t=Fault f and f: f ∈ F
        by auto
      from t exec-c1
      have Γ⊢⟨Seq c1 c2,s⟩ ⇒ Fault f
        by (auto intro: exec-Seq')
      with noabort f have False
        by (auto simp add: final-notin-def)
      hence False ..
    }
    ultimately show False by auto
  qed
qed

lemma Seq-NoFaultStuckD2:
  assumes noabort: Γ⊢⟨Seq c1 c2,s⟩ ⇒∉({Stuck} ∪ Fault ` F)
  shows ∀ t. Γ⊢⟨c1,s⟩ ⇒ t ⟶ t∉ ({Stuck} ∪ Fault ` F) ⟶
             Γ⊢⟨c2,t⟩ ⇒∉({Stuck} ∪ Fault ` F)
using noabort
by (auto simp add: final-notin-def intro: exec-Seq')


lemma MGT-implies-complete:
  assumes MGT: ∀ Z. Γ,{}⊢/F {s. s=Z ∧ Γ⊢⟨c,Normal s⟩ ⇒∉({Stuck} ∪ Fault
` (−F))} c
                           {t. Γ⊢⟨c,Normal Z⟩ ⇒ Normal t},
                           {t. Γ⊢⟨c,Normal Z⟩ ⇒ Abrupt t}
  assumes valid: Γ ⊨/F P c Q,A
  shows Γ,{} ⊢/F P c Q,A
  using MGT
  apply (rule ConseqMGT)
  apply (insert valid)
  apply (auto simp add: valid-def intro!: final-notinI)
  done
```

Equipped only with the classic consequence rule $[\![?\Gamma, ?\Theta \vdash_{/?F}\ ?P'\ ?c\ ?Q', ?A';\ ?P \subseteq ?P';\ ?Q' \subseteq ?Q;\ ?A' \subseteq ?A]\!] \Longrightarrow\ ?\Gamma, ?\Theta \vdash_{/?F}\ ?P\ ?c\ ?Q, ?A$ we can only derive this syntactically more involved version of completeness. But semantically it is equivalent to the "real" one (see below)

lemma *MGT-implies-complete'*:
assumes *MGT*: $\forall Z.\ \Gamma,\{\}\vdash_{/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup\ Fault\ `\ (-F))\}\ c$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
assumes *valid*: $\Gamma \models_{/F} P\ c\ Q,A$
shows $\Gamma,\{\} \vdash_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
using *MGT* [*rule-format*, *of Z*]
apply (*rule conseqPrePost*)
apply (*insert valid*)
apply (*fastforce simp add*: *valid-def final-notin-def*)
apply (*fastforce simp add*: *valid-def*)
apply (*fastforce simp add*: *valid-def*)
done

Semantic equivalence of both kind of formulations

lemma *valid-involved-to-valid*:
assumes *valid*:
$\forall Z.\ \Gamma\models_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
shows $\Gamma \models_{/F} P\ c\ Q,A$
using *valid*
apply (*simp add*: *valid-def*)
apply *clarsimp*
apply (*erule-tac x=x* **in** *allE*)
apply (*erule-tac x=Normal x* **in** *allE*)
apply (*erule-tac x=t* **in** *allE*)
apply *fastforce*
done

The sophisticated consequence rule allow us to do this semantical transformation on the hoare-level, too. The magic is, that it allow us to choose the instance of Z under the assumption of an state $s \in P$

lemma
assumes *deriv*:
$\forall Z.\ \Gamma,\{\}\vdash_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
shows $\Gamma,\{\} \vdash_{/F} P\ c\ Q,A$
apply (*rule ConseqMGT* [*OF deriv*])
apply *auto*
done

lemma *valid-to-valid-involved*:
$\Gamma \models_{/F} P\ c\ Q,A \Longrightarrow$
$\Gamma\models_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$

by (*simp add*: *valid-def Collect-conv-if*)

lemma
  assumes *deriv*: $\Gamma,\{\} \vdash_{/F} P\ c\ Q,A$
  shows $\Gamma,\{\}\vdash_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
  apply (*rule conseqPrePost* [*OF deriv*])
  apply *auto*
  done

lemma *conseq-extract-state-indep-prop*:
  assumes *state-indep-prop*:$\forall s \in P.\ R$
  assumes *to-show*: $R \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
  shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
  apply (*rule Conseq*)
  apply (*clarify*)
  apply (*rule-tac x=P* **in** *exI*)
  apply (*rule-tac x=Q* **in** *exI*)
  apply (*rule-tac x=A* **in** *exI*)
  using *state-indep-prop to-show*
  by *blast*

lemma *MGT-lemma*:
  assumes *MGT-Calls*:
    $\forall p{\in}dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{/F}$
      $\{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
       $(Call\ p)$
      $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
      $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  shows $\bigwedge Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$ c
      $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*induct c*)
  case *Skip*
  show $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Skip,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$ $Skip$
      $\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    by (*rule hoarep.Skip* [*THEN conseqPre*])
      (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
next
  case (*Basic f*)
  show $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Basic\ f,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$ $Basic\ f$
      $\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
      $\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    by (*rule hoarep.Basic* [*THEN conseqPre*])
      (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)

next
  case (*Spec r*)
  show $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
Spec r
    $\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
    $\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    apply (*rule hoarep.Spec* [*THEN conseqPre*])
    apply (*clarsimp simp add*: *final-notin-def*)
    apply (*case-tac* $\exists t.\ (Z,t) \in r$)
    apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
    done
next
  case (*Seq c1 c2*)
  have *hyp-c1*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$ *c1*
    $\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
    $\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *Seq.hyps* **by** *iprover*
  have *hyp-c2*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$ *c2*
    $\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
    $\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *Seq.hyps* **by** *iprover*
  from *hyp-c1*
  have $\Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
c1
    $\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t \wedge$
      $\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$,
    $\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    by (*rule ConseqMGT*)
      (*auto dest*: *Seq-NoFaultStuckD1* [*simplified*] *Seq-NoFaultStuckD2* [*simplified*]
        intro: *exec.Seq*)
  thus $\Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$

      Seq c1 c2
    $\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
    $\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  proof (*rule hoarep.Seq*)
    show $\Gamma,\Theta\vdash_{/F} \{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t \wedge$
        $\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
      c2
      $\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
      $\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
      fix *r t*
      assume $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ r$ $\Gamma\vdash\langle c2,Normal\ r\rangle \Rightarrow Normal\ t$
      then show $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t$
        by (*iprover intro*: *exec.intros*)
    next

```
      fix r t
      assume Γ⊢⟨c1,Normal Z⟩ ⇒ Normal r Γ⊢⟨c2,Normal r⟩ ⇒ Abrupt t
      then show Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Abrupt t
        by (iprover intro: exec.intros)
    qed
  qed
next
  case (Cond b c1 c2)
  have ∀ Z. Γ,Θ⊢/F{s. s=Z ∧ Γ⊢⟨c1,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
c1
                {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Normal t},
                {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Abrupt t}
    using Cond.hyps by iprover
  hence Γ,Θ⊢/F ({s. s=Z ∧ Γ⊢⟨Cond b c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))}∩b)
                  c1
               {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t}
    by (rule ConseqMGT)
       (fastforce intro: exec.CondTrue simp add: final-notin-def)
  moreover
  have ∀ Z. Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
c2
                  {t. Γ⊢⟨c2,Normal Z⟩ ⇒ Normal t},
                  {t. Γ⊢⟨c2,Normal Z⟩ ⇒ Abrupt t}
    using Cond.hyps by iprover
  hence Γ,Θ⊢/F({s. s=Z ∧ Γ⊢⟨Cond b c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))}∩−b)
                 c2
               {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t}
    by (rule ConseqMGT)
       (fastforce intro: exec.CondFalse simp add: final-notin-def)
  ultimately
  show Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨Cond b c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))}
                Cond b c1 c2
             {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t},
             {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t}
    by (rule hoarep.Cond)
next
  case (While b c)
  let ?unroll = ({(s,t). s∈b ∧ Γ⊢⟨c,Normal s⟩ ⇒ Normal t})*
  let ?P' = λZ. {t. (Z,t)∈?unroll ∧
                  (∀ e. (Z,e)∈?unroll ⟶ e∈b
                        ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                           (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                                Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u))}
  let ?A' = λZ. {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t}
```

```
show Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨While b c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ` (−F))}

           While b c
          {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t},
          {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t}
 proof (rule ConseqMGT [where ?P'=?P'
                  and ?Q'=λZ. ?P' Z ∩ − b and ?A'=?A'])
  show ∀ Z. Γ,Θ⊢/F (?P' Z) (While b c) (?P' Z ∩ − b),(?A' Z)
  proof (rule allI, rule hoarep.While)
    fix Z
    from While
    have ∀ Z. Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ` (−F))}
c
                  {t. Γ⊢⟨c,Normal Z⟩ ⇒ Normal t},
                  {t. Γ⊢⟨c,Normal Z⟩ ⇒ Abrupt t} by iprover
    then show Γ,Θ⊢/F (?P' Z ∩ b) c (?P' Z),(?A' Z)
    proof (rule ConseqMGT)
      fix s
      assume  s∈ {t. (Z, t) ∈ ?unroll ∧
                 (∀ e. (Z,e)∈?unroll ⟶ e∈b
                     ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                        (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                             Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u))}
              ∩ b
      then obtain
        Z-s-unroll: (Z,s) ∈ ?unroll and
        noabort:∀ e. (Z,e)∈?unroll ⟶ e∈b
                   ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                      (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                           Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u) and
        s-in-b: s∈b
        by blast
      show s ∈ {t. t = s ∧ Γ⊢⟨c,Normal t⟩ ⇒∉({Stuck} ∪ Fault ` (−F))} ∧
      (∀ t. t ∈ {t. Γ⊢⟨c,Normal s⟩ ⇒ Normal t} ⟶
          t ∈ {t. (Z, t) ∈ ?unroll ∧
               (∀ e. (Z,e)∈?unroll ⟶  e∈b
                   ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                      (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                           Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u))}) ∧
       (∀ t. t ∈ {t. Γ⊢⟨c,Normal s⟩ ⇒ Abrupt t} ⟶
          t ∈ {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t})
        (is ?C1 ∧ ?C2 ∧ ?C3)
      proof (intro conjI)
        from Z-s-unroll noabort s-in-b show ?C1 by blast
      next
        {
          fix t
          assume s-t: Γ⊢⟨c,Normal s⟩ ⇒ Normal t
          moreover
```

```
          from Z-s-unroll s-t s-in-b
          have (Z, t) ∈ ?unroll
            by (blast intro: rtrancl-into-rtrancl)
          moreover note noabort
          ultimately
          have (Z, t) ∈ ?unroll ∧
                (∀ e. (Z,e)∈?unroll ⟶ e∈b
                      ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                          (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                                Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u))
            by iprover
        }
        then show ?C2 by blast
      next
        {
          fix t
          assume s-t: Γ⊢⟨c,Normal s⟩ ⇒ Abrupt t
          from Z-s-unroll noabort s-t s-in-b
          have Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t
            by blast
        } thus ?C3 by simp
      qed
    qed
  qed
next
  fix s
   assume P: s ∈ {s. s=Z ∧ Γ⊢⟨While b c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))}
  hence WhileNoFault: Γ⊢⟨While b c,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
    by auto
  show s ∈ ?P' s ∧
  (∀ t. t∈(?P' s ∩ − b)⟶
       t∈{t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t})∧
  (∀ t. t∈?A' s ⟶ t∈?A' Z)
  proof (intro conjI)
    {
      fix e
      assume (Z,e) ∈ ?unroll e ∈ b
      from this WhileNoFault
      have Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
             (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                  Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u) (is ?Prop Z e)
      proof (induct rule: converse-rtrancl-induct [consumes 1])
        assume e-in-b: e ∈ b
         assume WhileNoFault: Γ⊢⟨While b c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))
        with e-in-b WhileNoFault
        have cNoFault: Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
          by (auto simp add: final-notin-def intro: exec.intros)
```

```
        moreover
        {
          fix u assume Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u
          with e-in-b have Γ⊢⟨While b c,Normal e⟩ ⇒ Abrupt u
            by (blast intro: exec.intros)
        }
        ultimately
        show ?Prop e e
          by iprover
      next
        fix Z r
        assume e-in-b: e∈b
         assume WhileNoFault: Γ⊢⟨While b c,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))
       assume hyp: [[e∈b;Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))]]
                   =⇒ ?Prop r e
        assume Z-r:
          (Z, r) ∈ {(Z, r). Z ∈ b ∧ Γ⊢⟨c,Normal Z⟩ ⇒ Normal r}
        with WhileNoFault
        have Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
          by (auto simp add: final-notin-def intro: exec.intros)
        from hyp [OF e-in-b this] obtain
          cNoFault: Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) and
          Abrupt-r: ∀ u. Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u −→
                        Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u
          by simp

         {
          fix u assume Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u
          with Abrupt-r have Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u by simp
          moreover from  Z-r obtain
            Z ∈ b  Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
            by simp
          ultimately have Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u
            by (blast intro: exec.intros)
        }
        with cNoFault show ?Prop Z e
          by iprover
      qed
    }
    with P show s ∈ ?P′ s
      by blast
  next
    {
      fix t
      assume termination: t ∉ b
      assume (Z, t) ∈ ?unroll
      hence Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
      proof (induct rule: converse-rtrancl-induct [consumes 1])
```

```
          from termination
          show Γ⊢⟨While b c,Normal t⟩ ⇒ Normal t
            by (blast intro: exec.WhileFalse)
        next
          fix Z r
          assume first-body:
                 (Z, r) ∈ {(s, t). s ∈ b ∧ Γ⊢⟨c,Normal s⟩ ⇒ Normal t}
          assume (r, t) ∈ ?unroll
          assume rest-loop: Γ⊢⟨While b c, Normal r⟩ ⇒ Normal t
          show Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
          proof −
            from first-body obtain
              Z ∈ b Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
              by fast
            moreover
            from rest-loop have
              Γ⊢⟨While b c,Normal r⟩ ⇒ Normal t
              by fast
            ultimately show Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
              by (rule exec.WhileTrue)
          qed
        qed
      }
      with P
      show (∀ t. t∈(?P' s ∩ − b)
            ⟶t∈{t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t})
        by blast
    next
      from P show ∀ t. t∈?A' s ⟶ t∈?A' Z by simp
    qed
  qed
next
  case (Call p)
  let ?P = {s. s=Z ∧ Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
  from noStuck-Call have ∀ s ∈ ?P. p ∈ dom Γ
    by (fastforce simp add: final-notin-def )
  then show Γ,Θ⊢/F ?P (Call p)
              {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
              {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t}
  proof (rule conseq-extract-state-indep-prop)
    assume p-definied: p ∈ dom Γ
    with MGT-Calls show
      Γ,Θ⊢/F{s. s=Z ∧
               Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
                (Call p)
               {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨Call  p,Normal Z⟩ ⇒ Abrupt t}
      by (auto)
  qed
```

next
  case (*DynCom c*)
  have *hyp*:
    $\bigwedge s'. \forall Z. \Gamma,\Theta\vdash_{/F}\{s. s = Z \wedge \Gamma\vdash\langle c\ s',Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$c\ s'$
      $\{t. \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Normal\ t\},\{t. \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *DynCom* **by** *simp*
  have *hyp'*:
  $\Gamma,\Theta\vdash_{/F}\{s. s = Z \wedge \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}\ c$
Z
        $\{t. \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t. \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle$
$\Rightarrow Abrupt\ t\}$
    by (*rule ConseqMGT* [*OF hyp*])
      (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
   show $\Gamma,\Theta\vdash_{/F}\{s. s = Z \wedge \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F))\}$
              DynCom c
            $\{t. \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
            $\{t. \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    apply (*rule hoarep.DynCom*)
    apply (*clarsimp*)
    apply (*rule hyp'* [*simplified*])
    done
next
  case (*Guard f g c*)
   have *hyp-c*: $\forall Z. \Gamma,\Theta\vdash_{/F}\ \{s. s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F))\}\ c$
                  $\{t. \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
                  $\{t. \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *Guard* **by** *iprover*
  show *?case*
  proof (*cases* $f \in F$)
    case *True*
    from *hyp-c*
    have $\Gamma,\Theta\vdash_{/F}$ $(g \cap \{s. s = Z \wedge$
                  $\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-\ F))\})$
           c
          $\{t. \Gamma\vdash\langle Guard\ f\ g\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
          $\{t. \Gamma\vdash\langle Guard\ f\ g\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
      apply (*rule ConseqMGT*)
      apply (*insert True*)
      apply (*auto simp add*: *final-notin-def intro*: *exec.intros*)
      done
    from *True this*
    show *?thesis*
      by (*rule conseqPre* [*OF Guarantee*]) *auto*
  next
    case *False*

from *hyp-c*
have $\Gamma,\Theta\vdash_{/F}$
$(g \cap \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Guard}\ f\ g\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\})$

c
$\{t.\ \Gamma\vdash\langle \mathit{Guard}\ f\ g\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Guard}\ f\ g\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
apply (*rule ConseqMGT*)
apply *clarify*
apply (*frule Guard-noFaultStuckD* [*OF - False*])
apply (*auto simp add*: *final-notin-def intro*: *exec.intros*)
done
then show *?thesis*
apply (*rule conseqPre* [*OF hoarep.Guard*])
apply *clarify*
apply (*frule Guard-noFaultStuckD* [*OF - False*])
apply *auto*
done
qed
next
case *Throw*
show $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
Throw
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule conseqPre* [*OF hoarep.Throw*]) (*blast intro*: *exec.intros*)
next
case (*Catch* c_1 c_2)
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
c_1
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F))\}$ c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t \wedge$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
c_2
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}\{s.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ s \wedge$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
c_2

$\{t.\ \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
ultimately
show $\Gamma,\Theta \vdash_{/F} \{s.\ s = Z \wedge \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$Catch\ c_1\ c_2$
$\{t.\ \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoarep.Catch*)
qed

lemma *MGT-Calls*:
$\forall p \in dom\ \Gamma.\ \forall Z.$
$\Gamma,\{\} \vdash_{/F} \{s.\ s=Z \wedge \Gamma \vdash \langle Call\ p, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$(Call\ p)$
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof −
{
fix $p\ Z$
assume *defined*: $p \in dom\ \Gamma$
have
$\Gamma,(\bigcup p \in dom\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s=Z \wedge$
$\Gamma \vdash \langle Call\ p, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F))\}$,
p,
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\})\})$
$\vdash_{/F} \{s.\ s = Z \wedge \Gamma \vdash \langle Call\ p, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
(**is** $\Gamma, ?\Theta \vdash_{/F} (?Pre\ p\ Z)\ (the\ (\Gamma\ p))\ (?Post\ p\ Z), (?Abr\ p\ Z)$)
proof −
have *MGT-Calls*:
$\forall p \in dom\ \Gamma.\ \forall Z.\ \Gamma, ?\Theta \vdash_{/F}$
$\{s.\ s=Z \wedge \Gamma \vdash \langle Call\ p, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$(Call\ p)$
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*intro ballI allI*, *rule HoarePartialDef.Asm*,*auto*)
have $\forall Z.\ \Gamma, ?\Theta \vdash_{/F} \{s.\ s=Z \wedge \Gamma \vdash \langle the\ (\Gamma\ p)\ , Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault`(-F))\}$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma \vdash \langle the\ (\Gamma\ p), Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle the\ (\Gamma\ p), Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

by (*iprover intro*: *MGT-lemma* [*OF MGT-Calls*])
thus $\Gamma,?\Theta\vdash_{/F}$ (*?Pre p Z*) (*the* (Γ *p*)) (*?Post p Z*),(*?Abr p Z*)
apply (*rule ConseqMGT*)
apply (*clarify*,*safe*)
proof −
assume $\Gamma\vdash\langle$*Call p*,*Normal Z*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))$
with *defined* **show** $\Gamma\vdash\langle$*the* (Γ *p*),*Normal Z*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))$
by (*fastforce simp add*: *final-notin-def*
intro: *exec.intros*)
next
fix *t*
assume $\Gamma\vdash\langle$*the* (Γ *p*),*Normal Z*$\rangle \Rightarrow$ *Normal t*
with *defined*
show $\Gamma\vdash\langle$*Call p*,*Normal Z*$\rangle \Rightarrow$*Normal t*
by (*auto intro*: *exec.Call*)
next
fix *t*
assume $\Gamma\vdash\langle$*the* (Γ *p*),*Normal Z*$\rangle \Rightarrow$ *Abrupt t*
with *defined*
show $\Gamma\vdash\langle$*Call p*,*Normal Z*$\rangle \Rightarrow$*Abrupt t*
by (*auto intro*: *exec.Call*)
qed
qed
}
then show *?thesis*
apply −
apply (*intro ballI allI*)
apply (*rule CallRec'* [**where** *Procs*=*dom* Γ **and**
$P=\lambda p\ Z.\ \{s.\ s=Z\ \wedge$
$\Gamma\vdash\langle$*Call p*,*Normal s*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))\}$**and**
$Q=\lambda p\ Z.$
$\{t.\ \Gamma\vdash\langle$*Call p*,*Normal Z*$\rangle \Rightarrow$ *Normal t*$\}$ **and**
$A=\lambda p\ Z.$
$\{t.\ \Gamma\vdash\langle$*Call p*,*Normal Z*$\rangle \Rightarrow$ *Abrupt t*$\}$])
apply *simp+*
done
qed

theorem *hoare-complete*: $\Gamma\models_{/F} P\ c\ Q,A \Longrightarrow \Gamma,\{\}\vdash_{/F} P\ c\ Q,A$
by (*iprover intro*: *MGT-implies-complete MGT-lemma* [*OF MGT-Calls*])

lemma *hoare-complete'*:
assumes *cvalid*: $\forall\, n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
proof (*cases* $\Gamma\models_{/F} P\ c\ Q,A$)
case *True*
hence $\Gamma,\{\}\vdash_{/F} P\ c\ Q,A$
by (*rule hoare-complete*)

thus Γ,$\Theta$$\vdash_{/F}$ *P c Q*,*A*
  by (*rule hoare-augment-context*) *simp*
next
  case *False*
  with *cvalid*
  show *?thesis*
    by (*rule ExFalso*)
qed

lemma *hoare-strip-*Γ:
  assumes *deriv*: Γ,{}$\vdash_{/F}$ *P p Q*,*A*
  assumes F': $F' \subseteq -F$
  shows *strip* F' Γ,{}$\vdash_{/F}$ *P p Q*,*A*
proof (*rule hoare-complete*)
  from *hoare-sound* [*OF deriv*] **have** $\Gamma$$\models_{/F}$ *P p Q*,*A*
    by (*simp add*: *cvalid-def*)
  from *this* F'
  show *strip* F' $\Gamma$$\models_{/F}$ *P p Q*,*A*
    by (*rule valid-to-valid-strip*)
qed

10.3 And Now: Some Useful Rules

10.3.1 Consequence

lemma *LiberalConseq-sound*:
fixes *F*::$'f$ *set*
assumes *cons*: $\forall s \in P.\ \forall (t::('s,'f)\ xstate).\ \exists P'\ Q'\ A'.\ (\forall n.\ \Gamma,\Theta \models n:_{/F}\ P'\ c\ Q',A') \wedge$
$((s \in P' \longrightarrow t \in Normal\ `\ Q' \cup Abrupt\ `\ A')$
$\longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A)$
shows Γ,$\Theta$$\models$*n*:$_{/F}$ *P c Q*,*A*
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*:$\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models n:_{/F}\ P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma \vdash \langle c,Normal\ s\rangle\ =n\Rightarrow\ t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof −
    from *P cons* **obtain** P' Q' A' **where**
      spec: $\forall n.\ \Gamma,\Theta \models n:_{/F}\ P'\ c\ Q',A'$ **and**
      adapt: $(s \in P' \longrightarrow t \in Normal\ `\ Q' \cup Abrupt\ `\ A')$
        $\longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A$
      apply −
      apply (*drule* (*1*) *bspec*)
      apply (*erule-tac x*=*t* **in** *allE*)

apply (*elim exE conjE*)
apply *iprover*
done
from *exec spec ctxt t-notin-F*
have $s \in P' \longrightarrow t \in Normal$ ‘ $Q' \cup Abrupt$ ‘ A'
by (*simp add*: *cnvalid-def nvalid-def*)
with *adapt* **show** *?thesis*
by *simp*
qed
qed

lemma *LiberalConseq*:
fixes F:: *'f set*
assumes *cons*: $\forall s \in P.\ \forall (t::('s,'f)\ xstate).\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F} P'\ c\ Q',A' \wedge$
$((s \in P' \longrightarrow t \in Normal$ ‘ $Q' \cup Abrupt$ ‘ $A')$
$\longrightarrow t \in Normal$ ‘ $Q \cup Abrupt$ ‘ $A)$
shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule LiberalConseq-sound*)
using *cons*
apply (*clarify*)
apply (*drule* (*1*) *bspec*)
apply (*erule-tac x=t* **in** *allE*)
apply *clarify*
apply (*rule-tac x=P'* **in** *exI*)
apply (*rule-tac x=Q'* **in** *exI*)
apply (*rule-tac x=A'* **in** *exI*)
apply (*rule conjI*)
apply (*blast intro*: *hoare-cnvalid*)
apply *assumption*
done

lemma $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F} P'\ c\ Q',A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$
$\Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule LiberalConseq*)
apply (*rule ballI*)
apply (*drule* (*1*) *bspec*)
apply *clarify*
apply (*rule-tac x=P'* **in** *exI*)
apply (*rule-tac x=Q'* **in** *exI*)
apply (*rule-tac x=A'* **in** *exI*)
apply *auto*
done

lemma
fixes F:: *'f set*
assumes *cons*: $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F} P'\ c\ Q',A' \wedge$

$(\forall (t::('s,'f)\ xstate).\ (s \in P' \longrightarrow t \in Normal\ `\ Q' \cup Abrupt\ `\ A')$
$\longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A)$

shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
apply (*rule Conseq*)
apply (*rule ballI*)
apply (*insert cons*)
apply (*drule* (*1*) *bspec*)
apply *clarify*
apply (*rule-tac* $x=P'$ **in** *exI*)
apply (*rule-tac* $x=Q'$ **in** *exI*)
apply (*rule-tac* $x=A'$ **in** *exI*)
apply (*rule conjI*)
apply *assumption*

oops

lemma *LiberalConseq'*:
fixes F:: *'f set*
assumes *cons*: $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F}\ P'\ c\ Q',A' \wedge$
$(\forall (t::('s,'f)\ xstate).\ (s \in P' \longrightarrow t \in Normal\ `\ Q' \cup Abrupt\ `\ A')$
$\longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
apply (*rule LiberalConseq*)
apply (*rule ballI*)
apply (*rule allI*)
apply (*insert cons*)
apply (*drule* (*1*) *bspec*)
apply *clarify*
apply (*rule-tac* $x=P'$ **in** *exI*)
apply (*rule-tac* $x=Q'$ **in** *exI*)
apply (*rule-tac* $x=A'$ **in** *exI*)
apply *iprover*
done

lemma *LiberalConseq''*:
fixes F:: *'f set*
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
assumes *cons*: $\forall s\ (t::('s,'f)\ xstate).$
$(\forall Z.\ s \in P'\ Z \longrightarrow t \in Normal\ `\ Q'\ Z \cup Abrupt\ `\ A'\ Z)$
$\longrightarrow (s \in P \longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
apply (*rule LiberalConseq*)
apply (*rule ballI*)
apply (*rule allI*)
apply (*insert cons*)
apply (*erule-tac* $x=s$ **in** *allE*)
apply (*erule-tac* $x=t$ **in** *allE*)
apply (*case-tac* $t \in Normal\ `\ Q \cup Abrupt\ `\ A$)

apply (*insert spec*)
apply *iprover*
apply *auto*
done

primrec *procs*:: $('s,'p,'f)$ *com* $\Rightarrow$ $'p$ *set*
where
procs Skip = {} |
procs (*Basic f*) = {} |
procs (*Seq* c_1 c_2) = (*procs* c_1 $\cup$ *procs* c_2) |
procs (*Cond b* c_1 c_2) = (*procs* c_1 $\cup$ *procs* c_2) |
procs (*While b c*) = *procs c* |
procs (*Call p*) = {*p*} |
procs (*DynCom c*) = ($\bigcup$ *s*. *procs* (*c s*)) |
procs (*Guard f g c*) = *procs c* |
procs Throw = {} |
procs (*Catch* c_1 c_2) = (*procs* c_1 $\cup$ *procs* c_2)

primrec *noSpec*:: $('s,'p,'f)$ *com* $\Rightarrow$ *bool*
where
noSpec Skip = *True* |
noSpec (*Basic f*) = *True* |
noSpec (*Spec r*) = *False* |
noSpec (*Seq* c_1 c_2) = (*noSpec* c_1 $\wedge$ *noSpec* c_2) |
noSpec (*Cond b* c_1 c_2) = (*noSpec* c_1 $\wedge$ *noSpec* c_2) |
noSpec (*While b c*) = *noSpec c* |
noSpec (*Call p*) = *True* |
noSpec (*DynCom c*) = ($\forall$ *s*. *noSpec* (*c s*)) |
noSpec (*Guard f g c*) = *noSpec c* |
noSpec Throw = *True* |
noSpec (*Catch* c_1 c_2) = (*noSpec* c_1 $\wedge$ *noSpec* c_2)

lemma *exec-noSpec-no-Stuck*:
 assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
 assumes *noSpec-c*: *noSpec c*
 assumes *noSpec*-Γ: $\forall p \in$ *dom* Γ. *noSpec* (*the* (Γ *p*))
 assumes *procs-subset*: *procs c* $\subseteq$ *dom* Γ
 assumes *procs-subset*-Γ: $\forall p \in$ *dom* Γ. *procs* (*the* (Γ *p*)) $\subseteq$ *dom* Γ
 assumes *s-no-Stuck*: *s*$\neq$*Stuck*
 shows *t*$\neq$*Stuck*
using *exec noSpec-c procs-subset s-no-Stuck* **proof** *induct*
  case (*Call p bdy s t*) **with** *noSpec*-Γ *procs-subset*-Γ **show** *?case*
    by (*auto dest*!: *bspec* [*of - - p*])
next
  case (*DynCom c s t*) **then show** *?case*
   by *auto blast*
qed *auto*

lemma *execn-noSpec-no-Stuck*:

```
 assumes exec: Γ⊢⟨c,s⟩ =n⇒ t
 assumes noSpec-c: noSpec c
 assumes noSpec-Γ: ∀ p ∈ dom Γ. noSpec (the (Γ p))
 assumes procs-subset: procs c ⊆ dom Γ
 assumes procs-subset-Γ: ∀ p ∈ dom Γ. procs (the (Γ p)) ⊆ dom Γ
 assumes s-no-Stuck: s≠Stuck
 shows t≠Stuck
using exec noSpec-c procs-subset s-no-Stuck proof induct
  case (Call p bdy n s t) with noSpec-Γ procs-subset-Γ show ?case
    by (auto dest!: bspec [of - - p])
next
  case (DynCom c s t) then show ?case
    by auto blast
qed auto

lemma LiberalConseq-noguards-nothrows-sound:
assumes spec: ∀ Z. ∀ n. Γ,Θ⊨n:/F (P' Z) c (Q' Z),(A' Z)
assumes cons: ∀ s t. (∀ Z. s ∈ P' Z ⟶ t ∈  Q' Z )
                 ⟶ (s ∈ P ⟶ t ∈ Q )
assumes noguards-c: noguards c
assumes noguards-Γ: ∀ p ∈ dom Γ. noguards (the (Γ p))
assumes nothrows-c: nothrows c
assumes nothrows-Γ: ∀ p ∈ dom Γ. nothrows (the (Γ p))
assumes noSpec-c: noSpec c
assumes noSpec-Γ: ∀ p ∈ dom Γ. noSpec (the (Γ p))
assumes procs-subset: procs c ⊆ dom Γ
assumes procs-subset-Γ: ∀ p ∈ dom Γ. procs (the (Γ p)) ⊆ dom Γ
shows Γ,Θ⊨n:/F P c Q,A
proof (rule cnvalidI)
  fix s t
  assume ctxt:∀ (P, p, Q, A)∈Θ. Γ⊨n:/F P (Call p) Q,A
  assume exec: Γ⊢⟨c,Normal s⟩ =n⇒ t
  assume P: s ∈ P
  assume t-notin-F: t ∉ Fault ‘ F
  show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
  proof −
    from execn-noguards-no-Fault [OF exec noguards-c noguards-Γ]
     execn-nothrows-no-Abrupt [OF exec nothrows-c nothrows-Γ ]
     execn-noSpec-no-Stuck [OF exec
            noSpec-c  noSpec-Γ procs-subset
      procs-subset-Γ]
    obtain t' where t: t=Normal t'
      by (cases t) auto
    with exec spec ctxt
    have (∀ Z. s ∈ P' Z ⟶ t' ∈  Q' Z)
      by (unfold  cnvalid-def nvalid-def) blast
    with cons P t show ?thesis
      by simp
  qed
```

qed

lemma *LiberalConseq-noguards-nothrows*:
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
assumes *cons*: $\forall s\ t.\ (\forall Z.\ s \in P'\ Z \longrightarrow t \in\ Q'\ Z\ )$
$\longrightarrow (s \in P \longrightarrow t \in Q\ )$
assumes *noguards-c*: *noguards c*
assumes *noguards-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{noguards}\ (\mathit{the}\ (\Gamma\ p))$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{nothrows}\ (\mathit{the}\ (\Gamma\ p))$
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{noSpec}\ (\mathit{the}\ (\Gamma\ p))$
assumes *procs-subset*: $\mathit{procs}\ c \subseteq \mathit{dom}\ \Gamma$
assumes *procs-subset-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{procs}\ (\mathit{the}\ (\Gamma\ p)) \subseteq \mathit{dom}\ \Gamma$
shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule LiberalConseq-noguards-nothrows-sound*
[*OF - cons noguards-c noguards-*Γ *nothrows-c nothrows-*Γ
noSpec-c noSpec-Γ
procs-subset procs-subset-Γ])
apply (*insert spec*)
apply (*intro allI*)
apply (*erule-tac x=Z* **in** *allE*)
by (*rule hoare-cnvalid*)

lemma
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\{s.\ s=\mathit{fst}\ Z \wedge P\ s\ (\mathit{snd}\ Z)\}\ c\ \{t.\ Q\ (\mathit{fst}\ Z)\ (\mathit{snd}\ Z)\ t\},\{\}$
assumes *noguards-c*: *noguards c*
assumes *noguards-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{noguards}\ (\mathit{the}\ (\Gamma\ p))$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{nothrows}\ (\mathit{the}\ (\Gamma\ p))$
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{noSpec}\ (\mathit{the}\ (\Gamma\ p))$
assumes *procs-subset*: $\mathit{procs}\ c \subseteq \mathit{dom}\ \Gamma$
assumes *procs-subset-*Γ: $\forall p \in \mathit{dom}\ \Gamma.\ \mathit{procs}\ (\mathit{the}\ (\Gamma\ p)) \subseteq \mathit{dom}\ \Gamma$
shows $\forall \sigma.\ \Gamma,\Theta\vdash_{/F}\{s.\ s=\sigma\}\ c\ \{t.\ \forall l.\ P\ \sigma\ l \longrightarrow Q\ \sigma\ l\ t\},\{\}$
apply (*rule allI*)
apply (*rule LiberalConseq-noguards-nothrows*
[*OF spec - noguards-c noguards-*Γ *nothrows-c nothrows-*Γ
noSpec-c noSpec-Γ
procs-subset procs-subset-Γ])
apply *auto*
done

10.3.2 Modify Return

lemma *ProcModifyReturn-sound*:
  assumes *valid-call*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ call\ init\ p\ return'\ c\ Q,A$
  assumes *valid-modif*:
    $\forall \sigma.\ \forall n.\ \Gamma,\Theta{\models}n{:}_{/UNIV}\ \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  assumes *ret-modif*:
    $\forall s\ t.\ t \in Modif\ (init\ s)$
        $\longrightarrow return'\ s\ t = return\ s\ t$
  assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s)$
        $\longrightarrow return'\ s\ t = return\ s\ t$
  shows $\Gamma,\Theta \models n{:}_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  then have *ctxt'*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/UNIV}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  assume *exec*: $\Gamma{\vdash}\langle call\ init\ p\ return\ c,Normal\ s\rangle = n{\Rightarrow}\ t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *exec*
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases rule*: *execn-call-Normal-elim*)
    fix *bdy m t'*
    assume *bdy*: $\Gamma\ p = Some\ bdy$
    assume *exec-body*: $\Gamma{\vdash}\langle bdy,Normal\ (init\ s)\rangle = m{\Rightarrow}\ Normal\ t'$
    assume *exec-c*: $\Gamma{\vdash}\langle c\ s\ t',Normal\ (return\ s\ t')\rangle = Suc\ m{\Rightarrow}\ t$
    assume *n*: $n = Suc\ m$
    from *exec-body n bdy*
    have $\Gamma{\vdash}\langle Call\ p,Normal\ (init\ s)\rangle = n{\Rightarrow}\ Normal\ t'$
      by (*auto simp add*: *intro*: *execn.Call*)
    from *cnvalidD* [*OF valid-modif* [*rule-format*, *of n init s*] *ctxt' this*] *P*
    have $t' \in Modif\ (init\ s)$
      by *auto*
    with *ret-modif* **have** $Normal\ (return'\ s\ t') =$
      $Normal\ (return\ s\ t')$
      by *simp*
    with *exec-body exec-c bdy n*
    have $\Gamma{\vdash}\langle call\ init\ p\ return'\ c,Normal\ s\rangle = n{\Rightarrow}\ t$
      by (*auto intro*: *execn-call*)
    from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this*] *P t-notin-F*
    show *?thesis*
      by *simp*
  next
    fix *bdy m t'*
    assume *bdy*: $\Gamma\ p = Some\ bdy$
    assume *exec-body*: $\Gamma{\vdash}\langle bdy,Normal\ (init\ s)\rangle = m{\Rightarrow}\ Abrupt\ t'$
    assume *n*: $n = Suc\ m$
    assume *t*: $t = Abrupt\ (return\ s\ t')$

also from *exec-body n bdy*
have $\Gamma\vdash\langle Call\ p,Normal\ (init\ s)\rangle\ =n\Rightarrow\ Abrupt\ t'$
by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif* [*rule-format*, *of n init s*] *ctxt' this*] *P*
have $t' \in ModifAbr\ (init\ s)$
by *auto*
with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
by *simp*
finally have $t = Abrupt\ (return'\ s\ t')$ **.**
with *exec-body bdy n*
have $\Gamma\vdash\langle call\ init\ p\ return'\ c,Normal\ s\rangle\ =n\Rightarrow\ t$
by (*auto intro*: *execn-callAbrupt*)
from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy m f*
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ =m\Rightarrow\ Fault\ f$ $n = Suc\ m$
$t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$
by (*auto intro*: *execn-callFault*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
next
fix *bdy m*
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ =m\Rightarrow\ Stuck$ $n = Suc\ m$
$t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$
by (*auto intro*: *execn-callStuck*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
next
fix *m*
assume $\Gamma\ p = None$
and $n = Suc\ m$ $t = Stuck$
then have $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$
by (*auto intro*: *execn-callUndefined*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
qed
qed

lemma *ProcModifyReturn*:

assumes *spec*: $\Gamma,\Theta\vdash_{/F}$ P (*call init p return′ c*) Q,A
assumes *result-conform*:
$\forall s\ t.\ t \in$ *Modif* (*init s*) $\longrightarrow$ (*return′ s t*) = (*return s t*)
assumes *return-conform*:
$\forall s\ t.\ t \in$ *ModifAbr* (*init s*)
$\longrightarrow$ (*return′ s t*) = (*return s t*)
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ *Call p* (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{/F}$ P (*call init p return c*) Q,A
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule ProcModifyReturn-sound*
[**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
OF - - result-conform return-conform])
using *spec*
apply (*blast intro*: *hoare-cnvalid*)
using *modifies-spec*
apply (*blast intro*: *hoare-cnvalid*)
done

lemma *ProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}$ P *call init p return′ c* Q,A
assumes *valid-modif*:
$\forall \sigma.\ \forall n.\ \Gamma,\Theta\models n{:}_{/F}$ $\{\sigma\}$ *Call p* (*Modif* σ),(*ModifAbr* σ)
assumes *ret-modif*:
$\forall s\ t.\ t \in$ *Modif* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
shows $\Gamma,\Theta \models n{:}_{/F}$ P (*call init p return c*) Q,A
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F}$ P (*Call p*) Q,A
assume *exec*: $\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
from *exec*
show $t \in$ *Normal* ‘ Q $\cup$ *Abrupt* ‘ A
proof (*cases rule*: *execn-call-Normal-elim*)
fix *bdy m t′*
assume *bdy*: Γ *p* = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle$ =*m*$\Rightarrow$ *Normal t′*
assume *exec-c*: $\Gamma\vdash\langle$*c s t′*,*Normal* (*return s t′*)$\rangle$ =*Suc m*$\Rightarrow$ *t*
assume *n*: *n* = *Suc m*
from *exec-body n bdy*
have $\Gamma\vdash\langle$*Call p*,*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Normal t′*
by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif* [*rule-format*, *of n init s*] *ctxt this*] *P*

have $t' \in$ *Modif* (*init s*)
by *auto*
with *ret-modif* **have** *Normal* (*return' s t'*) =
Normal (*return s t'*)
by *simp*
with *exec-body exec-c bdy n*
have Γ⊢⟨*call init p return' c*,*Normal s*⟩ =*n*⇒ *t*
by (*auto intro*: *execn-call*)
from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy m t'*
assume *bdy*: Γ *p* = *Some bdy*
assume *exec-body*: Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Abrupt t'*
assume *n*: *n* = *Suc m*
assume *t*: *t* = *Abrupt* (*return s t'*)
also
from *exec-body n bdy*
have Γ⊢⟨*Call p*,*Normal* (*init s*)⟩ =*n* ⇒ *Abrupt t'*
by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif* [*rule-format*, *of n init s*] *ctxt this*] *P*
have $t' \in$ *ModifAbr* (*init s*)
by *auto*
with *ret-modifAbr* **have** *Abrupt* (*return s t'*) = *Abrupt* (*return' s t'*)
by *simp*
finally have *t* = *Abrupt* (*return' s t'*) **.**
with *exec-body bdy n*
have Γ⊢⟨*call init p return' c*,*Normal s*⟩ =*n*⇒ *t*
by (*auto intro*: *execn-callAbrupt*)
from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy m f*
assume *bdy*: Γ *p* = *Some bdy*
assume Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Fault f n* = *Suc m* **and**
t: *t* = *Fault f*
with *bdy* **have** Γ⊢⟨*call init p return' c* ,*Normal s*⟩ =*n*⇒ *t*
by (*auto intro*: *execn-callFault*)
from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this P*] *t t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy m*
assume *bdy*: Γ *p* = *Some bdy*
assume Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Stuck n* = *Suc m*
t = *Stuck*
with *bdy* **have** Γ⊢⟨*call init p return' c* ,*Normal s*⟩ =*n*⇒ *t*

by (*auto intro*: *execn-callStuck*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
next
fix *m*
assume Γ *p* = *None*
and *n* = *Suc m t* = *Stuck*
then have $\Gamma\vdash\langle$*call init p return' c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
by (*auto intro*: *execn-callUndefined*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
qed
qed

lemma *ProcModifyReturnSameFaults*:
assumes *spec*: $\Gamma,\Theta\vdash_{/F}$ *P* (*call init p return' c*) *Q*,*A*
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *return-conform*:
$\forall s\ t.\ t \in ModifAbr\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*call init p return c*) *Q*,*A*
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule ProcModifyReturnSameFaults-sound*
[**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
OF - - *result-conform return-conform*])
using *spec*
apply (*blast intro*: *hoare-cnvalid*)
using *modifies-spec*
apply (*blast intro*: *hoare-cnvalid*)
done

10.3.3 DynCall

lemma *dynProcModifyReturn-sound*:
assumes *valid-call*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}$ *P dynCall init p return' c Q*,*A*
assumes *valid-modif*:
$\forall s \in P.\ \forall \sigma.\ \forall n.$
$\Gamma,\Theta\models n{:}_{/UNIV} \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
assumes *ret-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s)$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s)$
$\longrightarrow return'\ s\ t = return\ s\ t$

shows $\Gamma,\Theta \models n:_{/F}\ P\ (\mathit{dynCall\ init\ p\ return\ c})\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models n:_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
  then have *ctxt'*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models n:_{/UNIV}\ P\ (\mathit{Call}\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  assume *exec*: $\Gamma\vdash\langle \mathit{dynCall\ init\ p\ return\ c},\mathit{Normal}\ s\rangle =n\Rightarrow t$
  assume *t-notin-F*: $t \notin \mathit{Fault}$ ‘ F
  assume *P*: $s \in P$
  with *valid-modif*
  have *valid-modif'*: $\forall \sigma.\ \forall n.$
      $\Gamma,\Theta\models n:_{/UNIV}\ \{\sigma\}\ \mathit{Call}\ (p\ s)\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
    by *blast*
  from *exec*
  have $\Gamma\vdash\langle \mathit{call\ init}\ (p\ s)\ \mathit{return}\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$
    by (*cases rule*: *execn-dynCall-Normal-elim*)
  then show $t \in \mathit{Normal}$ ‘ $Q \cup \mathit{Abrupt}$ ‘ A
  proof (*cases rule*: *execn-call-Normal-elim*)
    fix *bdy m t'*
    assume *bdy*: $\Gamma\ (p\ s) = \mathit{Some\ bdy}$
    assume *exec-body*: $\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle =m\Rightarrow \mathit{Normal}\ t'$
    assume *exec-c*: $\Gamma\vdash\langle c\ s\ t',\mathit{Normal}\ (\mathit{return}\ s\ t')\rangle =\mathit{Suc}\ m\Rightarrow t$
    assume *n*: $n = \mathit{Suc}\ m$
    from *exec-body n bdy*
    have $\Gamma\vdash\langle \mathit{Call}\ (p\ s)\ ,\mathit{Normal}\ (\mathit{init}\ s)\rangle =n\Rightarrow \mathit{Normal}\ t'$
      by (*auto simp add*: *intro*: *execn.intros*)
    from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt' this*] *P*
    have $t' \in \mathit{Modif}\ (\mathit{init}\ s)$
      by *auto*
    with *ret-modif* **have** $\mathit{Normal}\ (\mathit{return}'\ s\ t') = \mathit{Normal}\ (\mathit{return}\ s\ t')$
      by *simp*
    with *exec-body exec-c bdy n*
    have $\Gamma\vdash\langle \mathit{call\ init}\ (p\ s)\ \mathit{return}'\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$
      by (*auto intro*: *execn-call*)
    hence $\Gamma\vdash\langle \mathit{dynCall\ init\ p\ return}'\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$
      by (*rule execn-dynCall*)
    from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
    show *?thesis*
      by *simp*
  next
    fix *bdy m t'*
    assume *bdy*: $\Gamma\ (p\ s) = \mathit{Some\ bdy}$
    assume *exec-body*: $\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle =m\Rightarrow \mathit{Abrupt}\ t'$
    assume *n*: $n = \mathit{Suc}\ m$
    assume *t*: $t = \mathit{Abrupt}\ (\mathit{return}\ s\ t')$
    also from *exec-body n bdy*
    have $\Gamma\vdash\langle \mathit{Call}\ (p\ s)\ ,\mathit{Normal}\ (\mathit{init}\ s)\rangle =n\Rightarrow \mathit{Abrupt}\ t'$
      by (*auto simp add*: *intro*: *execn.intros*)
    from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt' this*] *P*

have $t' \in$ *ModifAbr* (*init s*)
  by *auto*
with *ret-modifAbr* **have** *Abrupt* (*return s t'*) = *Abrupt* (*return' s t'*)
  by *simp*
finally have t = *Abrupt* (*return' s t'*) **.**
with *exec-body bdy n*
have $\Gamma\vdash\langle$*call init* (*p s*) *return' c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*auto intro*: *execn-callAbrupt*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m f*
assume *bdy*: Γ (*p s*) = *Some bdy*
assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle$ =*m*$\Rightarrow$ *Fault f n* = *Suc m*
  t = *Fault f*
with *bdy* **have** $\Gamma\vdash\langle$*call init* (*p s*) *return' c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*auto intro*: *execn-callFault*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
next
fix *bdy m*
assume *bdy*: Γ (*p s*) = *Some bdy*
assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle$ =*m*$\Rightarrow$ *Stuck n* = *Suc m*
  t = *Stuck*
with *bdy* **have** $\Gamma\vdash\langle$*call init* (*p s*) *return' c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*auto intro*: *execn-callStuck*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
next
fix *m*
assume Γ (*p s*) = *None*
and *n* = *Suc m t* = *Stuck*
hence $\Gamma\vdash\langle$*call init* (*p s*) *return' c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*auto intro*: *execn-callUndefined*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
qed

qed

lemma *dynProcModifyReturn*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{/F}$ *P dynCall init p return′ c Q*,*A*
assumes *ret-modif*:
  $\forall s\ t.\ t \in$ *Modif* (*init s*)
    $\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*)
    $\longrightarrow$ *return′ s t* = *return s t*
assumes *modif*:
  $\forall s \in P.\ \forall \sigma.$
    $\Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*dynCall init p return c*) *Q*,*A*
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule dynProcModifyReturn-sound* [**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
    OF hoare-cnvalid [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-cnvalid* [*OF modif* [*rule-format*]])
apply *assumption*
done

lemma *dynProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}$ *P dynCall init p return′ c Q*,*A*
assumes *valid-modif*:
  $\forall s \in P.\ \forall \sigma.\ \forall n.$
    $\Gamma,\Theta\models n{:}_{/F}$ $\{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
assumes *ret-modif*:
  $\forall s\ t.\ t \in$ *Modif* (*init s*) $\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*) $\longrightarrow$ *return′ s t* = *return s t*
shows $\Gamma,\Theta \models n{:}_{/F}$ *P* (*dynCall init p return c*) *Q*,*A*
proof (*rule cnvalidI*)
 fix *s t*
 assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models n{:}_{/F}$ *P* (*Call p*) *Q*,*A*
 assume *exec*: $\Gamma\vdash\langle$*dynCall init p return c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
 assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
 assume *P*: $s \in P$
 with *valid-modif*
 have *valid-modif′*: $\forall \sigma.\ \forall n.$
  $\Gamma,\Theta\models n{:}_{/F}$ $\{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
  by *blast*
 from *exec*
 have $\Gamma\vdash\langle$*call init* (*p s*) *return c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
  by (*cases rule*: *execn-dynCall-Normal-elim*)
 then show $t \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
 proof (*cases rule*: *execn-call-Normal-elim*)
  fix *bdy m t′*
  assume *bdy*: Γ (*p s*) = *Some bdy*

assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Normal\ t'$
assume *exec-c*: $\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle =Suc\ m\Rightarrow t$
assume *n*: $n = Suc\ m$
from *exec-body n bdy*
have $\Gamma\vdash\langle Call\ (p\ s)\ ,Normal\ (init\ s)\rangle =n \Rightarrow Normal\ t'$
  by (*auto simp add*: *intro*: *execn.Call*)
from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt this*] *P*
have $t' \in Modif\ (init\ s)$
  by *auto*
with *ret-modif* **have** $Normal\ (return'\ s\ t') = Normal\ (return\ s\ t')$
  by *simp*
with *exec-body exec-c bdy n*
have $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*auto intro*: *execn-call*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m t'*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Abrupt\ t'$
assume *n*: $n = Suc\ m$
assume *t*: $t = Abrupt\ (return\ s\ t')$
also from *exec-body n bdy*
have $\Gamma\vdash\langle Call\ (p\ s)\ ,Normal\ (init\ s)\rangle =n \Rightarrow Abrupt\ t'$
  by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt this*] *P*
have $t' \in ModifAbr\ (init\ s)$
  by *auto*
with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
  by *simp*
finally have $t = Abrupt\ (return'\ s\ t')$ **.**
with *exec-body bdy n*
have $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*auto intro*: *execn-callAbrupt*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m f*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Fault\ f\ n = Suc\ m$ **and**
  t: $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
  by (*auto intro*: *execn-callFault*)

hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Stuck\ n = Suc\ m$
  $t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
  by (*auto intro*: *execn-callStuck*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
next
fix *m*
assume $\Gamma\ (p\ s) = None$
and $n = Suc\ m\ t = Stuck$
hence $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
  by (*auto intro*: *execn-callUndefined*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
qed
qed

lemma *dynProcModifyReturnSameFaults*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{/F}\ P\ dynCall\ init\ p\ return'\ c\ Q,A$
assumes *ret-modif*:
  $\forall s\ t.\ t \in Modif\ (init\ s)$
    $\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s)$
    $\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif*:
  $\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule dynProcModifyReturnSameFaults-sound*
  [**where** *Modif=Modif* **and** *ModifAbr=ModifAbr*,
   OF hoare-cnvalid [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-cnvalid* [*OF modif* [*rule-format*]])
apply *assumption*

done

10.3.4 Conjunction of Postcondition

lemma *PostConjI-sound*:
assumes *valid-Q*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c\ Q,A$
assumes *valid-R*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c\ R,B$
shows $\Gamma,\Theta \models n{:}_{/F}\ P\ c\ (Q \cap R),(A \cap B)$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault$ ' F
  from *valid-Q* [*rule-format*] *ctxt exec P t-notin-F* **have** $t \in Normal$ ' $Q \cup Abrupt$ ' A
    by (*rule cnvalidD*)
  moreover
  from *valid-R* [*rule-format*] *ctxt exec P t-notin-F* **have** $t \in Normal$ ' $R \cup Abrupt$ ' B
    by (*rule cnvalidD*)
  ultimately show $t \in Normal$ ' $(Q \cap R) \cup Abrupt$ ' $(A \cap B)$
    by *blast*
qed

lemma *PostConjI*:
  assumes *deriv-Q*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
  assumes *deriv-R*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ R,B$
  shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ (Q \cap R),(A \cap B)$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule PostConjI-sound*)
using *deriv-Q*
apply (*blast intro*: *hoare-cnvalid*)
using *deriv-R*
apply (*blast intro*: *hoare-cnvalid*)
done

lemma *Merge-PostConj-sound*:
  assumes *validF*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
  assumes *validG*: $\forall n.\ \Gamma,\Theta\models n{:}_{/G}\ P'\ c\ R,X$
  assumes *F-G*: $F \subseteq G$
  assumes *P-P'*: $P \subseteq P'$
  shows $\Gamma,\Theta\models n{:}_{/F}\ P\ c\ (Q \cap R),(A \cap X)$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  with *F-G* **have** *ctxt'*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\models n{:}_{/G}\ P\ (Call\ p)\ Q,A$

by (*auto intro*: *nvalid-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
with *P-P′* **have** *P′*: $s \in P'$
by *auto*
assume *t-noFault*: $t \notin Fault$ ‘ F
show $t \in Normal$ ‘ $(Q \cap R) \cup Abrupt$ ‘ $(A \cap X)$
proof −
from *cnvalidD* [*OF validF* [*rule-format*] *ctxt exec P t-noFault*]
have $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A**.**
moreover from *this* **have** $t \notin Fault$ ‘ G
by *auto*
from *cnvalidD* [*OF validG* [*rule-format*] *ctxt′ exec P′ this*]
have $t \in Normal$ ‘ $R \cup Abrupt$ ‘ X **.**
ultimately show *?thesis* **by** *auto*
qed
qed

lemma *Merge-PostConj*:
assumes *validF*: $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
assumes *validG*: $\Gamma,\Theta\vdash_{/G} P'\ c\ R,X$
assumes *F-G*: $F \subseteq G$
assumes *P-P′*: $P \subseteq P'$
shows $\Gamma,\Theta\vdash_{/F} P\ c\ (Q \cap R),(A \cap X)$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule Merge-PostConj-sound* [*OF - - F-G P-P′*])
using *validF* **apply** (*blast intro*:*hoare-cnvalid*)
using *validG* **apply** (*blast intro*:*hoare-cnvalid*)
done

10.3.5 Weaken Context

lemma *WeakenContext-sound*:
assumes *valid-c*: $\forall n.\ \Gamma,\Theta'\models n{:}_{/F}\ P\ c\ Q,A$
assumes *valid-ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta'.\ \Gamma,\Theta\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
shows $\Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
with *valid-ctxt*
have *ctxt′*: $\forall (P,\ p,\ Q,\ A)\in\Theta'.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
by (*simp add*: *cnvalid-def*)
assume *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault$ ‘ F
from *valid-c* [*rule-format*] *ctxt′ exec P t-notin-F*
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A

    by (*rule cnvalidD*)
qed

lemma *WeakenContext*:
  assumes *deriv-c*: $\Gamma,\Theta'\vdash_{/F} P\ c\ Q,A$
  assumes *deriv-ctxt*: $\forall (P,p,Q,A)\in\Theta'.\ \Gamma,\Theta\vdash_{/F} P\ (Call\ p)\ Q,A$
  shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule WeakenContext-sound*)
using *deriv-c*
apply (*blast intro*: *hoare-cnvalid*)
using *deriv-ctxt*
apply (*blast intro*: *hoare-cnvalid*)
done

10.3.6 Guards and Guarantees

lemma *SplitGuards-sound*:
assumes *valid-c1*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F} P\ c_1\ Q,A$
assumes *valid-c2*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F} P\ c_2\ UNIV,UNIV$
assumes *c*: $(c_1 \cap_g c_2) = Some\ c$
shows $\Gamma,\Theta\models n{:}_{/F} P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault$ ‘ F
  show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
  proof (*cases t*)
    case *Normal*
    with *inter-guards-execn-noFault* [*OF c exec*]
    have $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
    from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
    show *?thesis*
      by (*rule cnvalidD*)
  next
    case *Abrupt*
    with *inter-guards-execn-noFault* [*OF c exec*]
    have $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
    from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
    show *?thesis*
      by (*rule cnvalidD*)
  next
    case (*Fault f*)
    with *exec inter-guards-execn-Fault* [*OF c*]
    have $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle c_2,Normal\ s\rangle =n\Rightarrow Fault\ f$

by *auto*
then show *?thesis*
proof (*cases rule*: *disjE* [*consumes 1*])
assume $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow Fault\ f$
from *Fault cnvalidD* [*OF valid-c1* [*rule-format*] *ctxt this P*] *t-notin-F*
show *?thesis*
by *blast*
next
assume $\Gamma\vdash\langle c_2,Normal\ s\rangle =n\Rightarrow Fault\ f$
from *Fault cnvalidD* [*OF valid-c2* [*rule-format*] *ctxt this P*] *t-notin-F*
show *?thesis*
by *blast*
qed
next
case *Stuck*
with *inter-guards-execn-noFault* [*OF c exec*]
have $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
qed
qed

lemma *SplitGuards*:
assumes *c*: $(c_1 \cap_g c_2) = Some\ c$
assumes *deriv-c1*: $\Gamma,\Theta\vdash_{/F} P\ c_1\ Q,A$
assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F} P\ c_2\ UNIV,UNIV$
shows $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule SplitGuards-sound* [*OF - - c*])
using *deriv-c1*
apply (*blast intro*: *hoare-cnvalid*)
using *deriv-c2*
apply (*blast intro*: *hoare-cnvalid*)
done

lemma *CombineStrip-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
assumes *valid-strip*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}\ P\ (strip\text{-}guards\ (-F)\ c)\ UNIV,UNIV$
shows $\Gamma,\Theta\models n{:}_{/\{\}}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}\ P\ (Call\ p)\ Q,A$
hence *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
by (*auto intro*: *nvalid-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$

```
  assume t-noFault: t ∉ Fault ‘ {}
  show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
  proof (cases t)
    case (Normal t')
    from cnvalidD [OF valid [rule-format] ctxt' exec P] Normal
    show ?thesis
      by auto
  next
    case (Abrupt t')
    from cnvalidD [OF valid [rule-format] ctxt' exec P] Abrupt
    show ?thesis
      by auto
  next
    case (Fault f)
    show ?thesis
    proof (cases f ∈ F)
      case True
      hence f ∉ −F by simp
      with exec Fault
      have Γ⊢⟨strip-guards (−F) c,Normal s⟩ =n⇒ Fault f
        by (auto intro: execn-to-execn-strip-guards-Fault)
      from cnvalidD [OF valid-strip [rule-format] ctxt this P] Fault
      have False
        by auto
      thus ?thesis ..
    next
      case False
      with cnvalidD [OF valid [rule-format] ctxt' exec P] Fault
      show ?thesis
        by auto
    qed
  next
    case Stuck
    from cnvalidD [OF valid [rule-format] ctxt' exec P] Stuck
    show ?thesis
      by auto
  qed
qed

lemma CombineStrip:
  assumes deriv: Γ,Θ⊢/F P c Q,A
  assumes deriv-strip: Γ,Θ⊢/{} P (strip-guards (−F) c) UNIV,UNIV
  shows Γ,Θ⊢/{} P c Q,A
apply (rule hoare-complete')
apply (rule allI)
apply (rule CombineStrip-sound)
apply  (iprover intro: hoare-cnvalid [OF deriv])
apply (iprover intro: hoare-cnvalid [OF deriv-strip])
done
```

lemma *GuardsFlip-sound*:
  assumes *valid*: $\forall\, n.\ \Gamma,\Theta{\models}n{:}_{/F}\ P\ c\ Q,A$
  assumes *validFlip*: $\forall\, n.\ \Gamma,\Theta{\models}n{:}_{/-F}\ P\ c\ UNIV,UNIV$
  shows $\Gamma,\Theta{\models}n{:}_{/\{\}}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall\,(P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}n{:}_{/\{\}}\ P\ (Call\ p)\ Q,A$
  hence *ctxt'*: $\forall\,(P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}n{:}_{/F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  from *ctxt* **have** *ctxtFlip*: $\forall\,(P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}n{:}_{/-F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle\ ={n}\Rightarrow\ t$
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin Fault\ `\ \{\}$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases t*)
    case (*Normal t'*)
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Normal*
    show *?thesis*
      by *auto*
  next
    case (*Abrupt t'*)
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Abrupt*
    show *?thesis*
      by *auto*
  next
    case (*Fault f*)
    show *?thesis*
    proof (*cases* $f \in F$)
      case *True*
      hence $f \notin -F$ **by** *simp*
      with *cnvalidD* [*OF validFlip* [*rule-format*] *ctxtFlip exec P*] *Fault*
      have *False*
        by *auto*
      thus *?thesis* **..**
    next
      case *False*
      with *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Fault*
      show *?thesis*
        by *auto*
    qed
  next
    case *Stuck*
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Stuck*
    show *?thesis*
      by *auto*
  qed

qed

lemma *GuardsFlip*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ *P c Q*,*A*
  assumes *derivFlip*: $\Gamma,\Theta\vdash_{/-F}$ *P c UNIV*,*UNIV*
  shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q*,*A*
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule GuardsFlip-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
apply (*iprover intro*: *hoare-cnvalid* [*OF derivFlip*])
done

lemma *MarkGuardsI-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}$ *P c Q*,*A*
  shows $\Gamma,\Theta\models n{:}_{/\{\}}$ *P mark-guards f c Q*,*A*
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}$ *P* (*Call p*) *Q*,*A*
  assume *exec*: $\Gamma\vdash\langle$*mark-guards f c*,*Normal s*$\rangle$ $=n\Rightarrow$ *t*
  from *execn-mark-guards-to-execn* [*OF exec*] **obtain** t' **where**
    exec-c: $\Gamma\vdash\langle c,$*Normal s*$\rangle$ $=n\Rightarrow t'$ **and**
    t'*-noFault*: $\neg$ *isFault* $t' \longrightarrow t' = t$
    by *blast*
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin$ *Fault* ‘ {}
  show $t \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
  proof −
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-c P*]
    have $t' \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
      by *blast*
    with t'*-noFault*
    show *?thesis*
      by *auto*
  qed
qed

lemma *MarkGuardsI*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q*,*A*
  shows $\Gamma,\Theta\vdash_{/\{\}}$ *P mark-guards f c Q*,*A*
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MarkGuardsI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MarkGuardsD-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}$ *P mark-guards f c Q*,*A*

shows $\Gamma,\Theta\models n:_{/\{\}}$ *P c Q*,*A*
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n:_{/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ‘ {}
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ *A*
proof (*cases isFault t*)
case *True*
with *execn-to-execn-mark-guards-Fault* [*OF exec*]
obtain *f'* **where** $\Gamma\vdash\langle mark\text{-}guards\ f\ c, Normal\ s\rangle =n\Rightarrow Fault\ f'$
by (*fastforce elim*: *isFaultE*)
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt this P*]
have *False*
by *auto*
thus *?thesis* **..**
next
case *False*
from *execn-to-execn-mark-guards* [*OF exec False*]
obtain *f'* **where** $\Gamma\vdash\langle mark\text{-}guards\ f\ c, Normal\ s\rangle =n\Rightarrow t$
by *auto*
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt this P*]
show *?thesis*
by *auto*
qed
qed

lemma *MarkGuardsD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/\{\}}$ *P mark-guards f c Q*,*A*
shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q*,*A*
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MarkGuardsD-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MergeGuardsI-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n:_{/F}$ *P c Q*,*A*
shows $\Gamma,\Theta\models n:_{/F}$ *P merge-guards c Q*,*A*
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n:_{/F}$ *P* (*Call p*) *Q*,*A*
assume *exec-merge*: $\Gamma\vdash\langle merge\text{-}guards\ c, Normal\ s\rangle =n\Rightarrow t$
from *execn-merge-guards-to-execn* [*OF exec-merge*]
have *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$ **.**
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*

from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec P t-notin-F*]
show $t \in$ *Normal* ' $Q \cup$ *Abrupt* ' A**.**
qed

lemma *MergeGuardsI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ P c Q,A
shows $\Gamma,\Theta\vdash_{/F}$ P *merge-guards* c Q,A
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MergeGuardsI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MergeGuardsD-sound*:
assumes *valid*: $\forall n$. $\Gamma,\Theta\models n$:$_{/F}$ P *merge-guards* c Q,A
shows $\Gamma,\Theta\models n$:$_{/F}$ P c Q,A
proof (*rule cnvalidI*)
fix s t
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta$. $\Gamma\models n$:$_{/F}$ P (*Call* p) Q,A
assume *exec*: $\Gamma\vdash\langle c$,*Normal* $s\rangle$ $=n\Rightarrow$ t
from *execn-to-execn-merge-guards* [*OF exec*]
have *exec-merge*: $\Gamma\vdash\langle$*merge-guards* c,*Normal* $s\rangle$ $=n\Rightarrow$ t**.**
assume P: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ' F
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-merge P t-notin-F*]
show $t \in$ *Normal* ' $Q \cup$ *Abrupt* ' A**.**
qed

lemma *MergeGuardsD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ P *merge-guards* c Q,A
shows $\Gamma,\Theta\vdash_{/F}$ P c Q,A
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MergeGuardsD-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *SubsetGuards-sound*:
assumes *c-c'*: $c \subseteq_g c'$
assumes *valid*: $\forall n$. $\Gamma,\Theta\models n$:$_{/\{\}}$ P c' Q,A
shows $\Gamma,\Theta\models n$:$_{/\{\}}$ P c Q,A
proof (*rule cnvalidI*)
fix s t
assume *ctxt*: $\forall (P, p, Q, A) \in \Theta$. $\Gamma\models n$:$_{/\{\}}$ P (*Call* p) Q,A
assume *exec*: $\Gamma\vdash\langle c$,*Normal* $s\rangle$ $=n\Rightarrow$ t
from *execn-to-execn-subseteq-guards* [*OF c-c' exec*] **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c'$,*Normal* $s\rangle$ $=n\Rightarrow$ t' **and**

t'-*noFault*: $\neg$ *isFault* $t' \longrightarrow t' = t$
    by *blast*
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin$ *Fault* ‘ {}
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-c′ P*] *t′-noFault t-noFault*
  show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
    by *auto*
qed

lemma *SubsetGuards*:
  assumes *c-c′*: $c \subseteq_g c'$
  assumes *deriv*: $\Gamma,\Theta\vdash_{/\{\}}$ $P\ c'\ Q,A$
  shows $\Gamma,\Theta\vdash_{/\{\}}$ $P\ c\ Q,A$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule SubsetGuards-sound* [*OF c-c′*])
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *NormalizeD-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}$ P (*normalize* c) Q,A
  shows $\Gamma,\Theta\models n{:}_{/F}$ $P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix $s\ t$
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/F}$ P (*Call* p) Q,A
  assume *exec*: $\Gamma\vdash\langle c,$*Normal* $s\rangle$ $=n\Rightarrow$ t
  hence *exec-norm*: $\Gamma\vdash\langle$*normalize* $c,$*Normal* $s\rangle$ $=n\Rightarrow$ t
    by (*rule execn-to-execn-normalize*)
  assume *P*: $s \in P$
  assume *noFault*: $t \notin$ *Fault* ‘ F
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-norm P noFault*]
  show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A**.**
qed

lemma *NormalizeD*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ P (*normalize* c) Q,A
  shows $\Gamma,\Theta\vdash_{/F}$ $P\ c\ Q,A$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule NormalizeD-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *NormalizeI-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}$ $P\ c\ Q,A$
  shows $\Gamma,\Theta\models n{:}_{/F}$ P (*normalize* c) Q,A
proof (*rule cnvalidI*)
  fix $s\ t$

  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  assume $\Gamma\vdash\langle normalize\ c, Normal\ s\rangle =n\Rightarrow t$
  hence *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$
    by (*rule execn-normalize-to-execn*)
  assume *P*: $s \in P$
  assume *noFault*: $t \notin Fault\ `\ F$
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec P noFault*]
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$**.**
qed

lemma *NormalizeI*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
  shows $\Gamma,\Theta\vdash_{/F}\ P\ (normalize\ c)\ Q,A$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule NormalizeI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

10.3.7 Restricting the Procedure Environment

lemma *nvalid-restrict-to-nvalid*:
assumes *valid-c*: $\Gamma|_M{\models}n{:}_{/F}\ P\ c\ Q,A$
shows $\Gamma{\models}n{:}_{/F}\ P\ c\ Q,A$
proof (*rule nvalidI*)
  fix *s t*
  assume *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof −
    from *execn-to-execn-restrict* [*OF exec*]
    obtain t' **where**
      exec-res: $\Gamma|_M\vdash\langle c, Normal\ s\rangle =n\Rightarrow t'$ **and**
      t-Fault: $\forall f.\ t = Fault\ f \longrightarrow t' \in \{Fault\ f,\ Stuck\}$ **and**
      t'*-notStuck*: $t'{\neq}Stuck \longrightarrow t'{=}t$
      by *blast*
    from *t-Fault t-notin-F* t'*-notStuck* **have** $t' \notin Fault\ `\ F$
      by (*cases* t') *auto*
    with *valid-c exec-res P*
    have $t' \in Normal\ `\ Q \cup Abrupt\ `\ A$
      by (*auto simp add*: *nvalid-def*)
    with t'*-notStuck*
    show *?thesis*
      by *auto*
  qed
qed

lemma *valid-restrict-to-valid*:

assumes *valid-c*: $\Gamma|_M\models_{/F}$ *P c Q*,*A*
shows $\Gamma\models_{/F}$ *P c Q*,*A*
proof (*rule validI*)
  fix *s t*
  assume *exec*: $\Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof −
    from *exec-to-exec-restrict* [*OF exec*]
    obtain t' **where**
      exec-res: $\Gamma|_M\vdash\langle c, Normal\ s\rangle \Rightarrow t'$ **and**
      t-Fault: $\forall f.\ t = Fault\ f \longrightarrow t' \in \{Fault\ f,\ Stuck\}$ **and**
      t'*-notStuck*: $t'\neq Stuck \longrightarrow t'=t$
      by *blast*
    from *t-Fault t-notin-F* t'*-notStuck* **have** $t' \notin Fault\ `\ F$
      by (*cases* t') *auto*
    with *valid-c exec-res P*
    have $t' \in Normal\ `\ Q \cup Abrupt\ `\ A$
      by (*auto simp add*: *valid-def*)
    with t'*-notStuck*
    show *?thesis*
      by *auto*
  qed
qed

lemma *augment-procs*:
assumes *deriv-c*: $\Gamma|_M,\{\}\vdash_{/F}$ *P c Q*,*A*
shows $\Gamma,\{\}\vdash_{/F}$ *P c Q*,*A*
  apply (*rule hoare-complete*)
  apply (*rule valid-restrict-to-valid*)
  apply (*insert hoare-sound* [*OF deriv-c*])
  by (*simp add*: *cvalid-def*)

lemma *augment-Faults*:
assumes *deriv-c*: $\Gamma,\{\}\vdash_{/F}$ *P c Q*,*A*
assumes *F*: $F \subseteq F'$
shows $\Gamma,\{\}\vdash_{/F'}$ *P c Q*,*A*
  apply (*rule hoare-complete*)
  apply (*rule valid-augment-Faults* [*OF - F*])
  apply (*insert hoare-sound* [*OF deriv-c*])
  by (*simp add*: *cvalid-def*)

end

theory *LocalRG-HoareDef*

imports *SmallStepCon EmbSimpl/HoarePartialProps HOL−Library.Countable*
begin

11 Validity of Correctness Formulas

11.1 Aux

abbreviation (*input*)
  set-fun :: *'a set* ⇒ *'a* ⇒ *bool* (-$_f$) **where**
  set-fun s ≡ λ*v*. *v*∈*s*

abbreviation (*input*)
  fun-set :: (*'a* ⇒ *bool*) ⇒ *'a set* (-$_s$) **where**
  fun-set f ≡ {σ. *f* σ}

lemma *tl-pair*:*Suc* (*Suc j*) < *length l* ⟹
  l1 = *tl l* ⟹
  P (*l*!(*Suc j*)) (*l*!(*Suc* (*Suc j*)))=
  P (*l1*!*j*) (*l1*!(*Suc j*))
by (*simp add*: *tl-zero-eq*)

lemma *for-all-k-sublist*:
assumes *a0*:*Suc* (*Suc j*)<*length l* **and**
  a1:(∀ *k* < *j*. *P* ((*tl l*)!*k*) ((*tl l*)!(*Suc k*))) **and**
  a2:*P* (*l*!*0*) (*l*!(*Suc 0*))
shows (∀ *k* < *Suc j*. *P* (*l*!*k*) (*l*!(*Suc k*)))
proof −
  {**fix** *k*
   assume *aa0*:*k* < *Suc j*
   have *P* (*l*!*k*) (*l*!(*Suc k*))
   proof (*cases k*)
     case *0* **thus** *?thesis* **using** *a2* **by** *auto*
   next
     case (*Suc k1*) **thus** *?thesis* **using** *aa0 a0 a1 a2*
     by (*metis SmallStepCon.nth-tl Suc-less-SucD dual-order.strict-trans length-greater-0-conv nth-Cons-Suc zero-less-Suc*)
   qed
  } **thus** *?thesis* **by** *auto*
qed

11.2 Validity for Component Programs.

type-synonym (*'s*,*'f*) *tran* = (*'s*,*'f*) *xstate* × (*'s*,*'f*) *xstate*
type-synonym (*'s*,*'p*,*'f*,*'e*) *rgformula* =
  ((*'s*,*'p*,*'f*,*'e*) *com* ×
   (*'s set*) ×
   ((*'s*,*'f*) *tran*) *set* ×
   ((*'s*,*'f*) *tran*) *set* ×

$('s\ set) \times$
$('s\ set))$

type-synonym $('s,'p,'f,'e)$ *sextuple* =
$('p \times$
$('s\ set) \times$
$(('s,'f)\ tran)\ set \times$
$(('s,'f)\ tran)\ set \times$
$('s\ set) \times$
$('s\ set))$

definition *Sta* :: $'s\ set \Rightarrow (('s,'f)\ tran)\ set \Rightarrow bool$ **where**
$Sta \equiv \lambda f\ g.\ (\forall x\ y\ x'.\ x' \in f \wedge x{=}Normal\ x' \longrightarrow\ (x,y){\in}\ g \longrightarrow (\exists y'.\ y{=}Normal\ y' \wedge y' \in f))$

lemma *Sta-intro*:$Sta\ a\ R \Longrightarrow Sta\ b\ R \Longrightarrow Sta\ (a \cap b)\ R$
unfolding *Sta-def* **by** *fastforce*

lemma *Sta-assoc*:$Sta\ (a \cap (b \cap c))\ R = Sta\ ((a{\cap}b){\cap}c)\ R$
unfolding *Sta-def* **by** *fastforce*

lemma *Sta-comm*:$Sta\ (a \cap b)\ R = Sta\ (b \cap a)\ R$
unfolding *Sta-def* **by** *fastforce*

lemma *Sta-add*:$Sta\ (a \cap b)\ R \Longrightarrow Sta\ (a \cap c)\ R \Longrightarrow$
$Sta\ (a \cap b \cap c)\ R$
unfolding *Sta-def* **by** *fastforce*

lemma *Sta-tran*:$Sta\ a\ R \Longrightarrow a = b \Longrightarrow Sta\ b\ R$
by *auto*

definition *Norm*:: $(('s,'f)\ tran)\ set \Rightarrow bool$ **where**
$Norm \equiv \lambda g.\ (\forall x\ y.\ (x,\ y) \in g \longrightarrow (\exists x'\ y'.\ x{=}Normal\ x' \wedge y{=}Normal\ y'))$

definition *env-tran*::
$('p \Rightarrow ('s,\ 'p,\ 'f,'e)\ LanguageCon.com\ option)$
$\Rightarrow ('s\ set)$
$\Rightarrow (('s,\ 'p,\ 'f,'e)\ LanguageCon.com \times ('s,\ 'f)\ xstate)\ list$
$\Rightarrow ('s,'f)\ tran\ set \Rightarrow bool$
where
$env\text{-}tran\ \Gamma\ q\ l\ rely \equiv snd(l!0) \in Normal\ `\ q \wedge (\forall i.\ Suc\ i{<}length\ l \longrightarrow$
$\Gamma{\vdash}_c(l!i)\ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in rely)$

definition *env-tran-right*::
$('p \Rightarrow ('s,\ 'p,\ 'f,'e)\ LanguageCon.com\ option)$
$\Rightarrow (('s,\ 'p,\ 'f,'e)\ LanguageCon.com \times ('s,\ 'f)\ xstate)\ list$
$\Rightarrow ('s,'f)\ tran\ set \Rightarrow bool$

where
env-tran-right Γ *l rely* $\equiv$
  $(\forall\, i.\ Suc\ i{<}length\ l \longrightarrow$
    $\Gamma\vdash_c(l!i)\ \ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
    $(snd(l!i),\ snd(l!(Suc\ i))) \in rely)$

lemma *env-tran-tail*:*env-tran-right* Γ $(x\#l)$ $R \Longrightarrow$ *env-tran-right* Γ *l R*
unfolding *env-tran-right-def*
by *fastforce*

lemma *env-tran-subr*:
assumes *a0*:*env-tran-right* Γ (*l1*@*l2*) *R*
shows *env-tran-right* Γ *l1 R*
unfolding *env-tran-right-def*
proof −
  {**fix** *i*
  assume *a1*:*Suc i*< *length l1*
  assume *a2*:$\Gamma\vdash_c$ *l1* ! *i* $\rightarrow_e$ *l1* ! *Suc i*
  then have *Suc i* < *length* (*l1*@*l2*) **using** *a1* **by** *fastforce*
  also then have $\Gamma\vdash_c$ (*l1*@*l2*) ! *i* $\rightarrow_e$ (*l1*@*l2*) ! *Suc i*
  proof −
    show *?thesis*
      by (*simp add*: *Suc-lessD a1 a2 nth-append*)
  qed
  ultimately have *f1*:(*snd* ((*l1*@*l2*)! *i*), *snd* ((*l1*@*l2*) ! *Suc i*)) $\in$ *R*
  using *a0* **unfolding** *env-tran-right-def* **by** *auto*
  then have (*snd* (*l1*! *i*), *snd* (*l1* ! *Suc i*)) $\in$ *R*
  using *a1*
proof −
  have $\forall$ *ps psa n*. *if n* < *length ps then* (*ps* @ *psa*) ! *n* = (*ps* ! *n*::('*b*, '*a*, '*c*,'*d*) *LanguageCon.com* × ('*b*, '*c*) *xstate*)
          else (*ps* @ *psa*) ! *n* = *psa* ! (*n* − *length ps*)
    by (*meson nth-append*)
  then show *?thesis*
    using *f1* ‹*Suc i* < *length l1*› **by** *force*
qed
  } **then show**
  $\forall$ *i*. *Suc i* < *length l1* $\longrightarrow$
    $\Gamma\vdash_c$ *l1* ! *i* $\rightarrow_e$ *l1* ! *Suc i* $\longrightarrow$
    (*snd* (*l1* ! *i*), *snd* (*l1* ! *Suc i*)) $\in$ *R*
  by *blast*
qed

definition *Satis* **where** *Satis* $\equiv$ *True*

definition *sep-conj* **where** *sep-conj* $\equiv$ *True*

lemma *env-tran-subl*:*env-tran-right* Γ (*l1*@*l2*) *R* $\Longrightarrow$ *env-tran-right* Γ *l2 R*

proof (*induct l1*)
  case *Nil* **thus** *?case* **by** *auto*
next
  case (*Cons a l1*) **thus** *?case* **by** (*fastforce intro*:*append-Cons env-tran-tail*)
qed

lemma *env-tran-R-R'*:*env-tran-right* Γ *l R* $\Longrightarrow$
    ($R \subseteq R'$) $\Longrightarrow$
    env-tran-right Γ *l R'*
unfolding *env-tran-right-def Satis-def sep-conj-def*
apply *clarify*
apply (*erule allE*)
apply *auto*
done

lemma *env-tran-normal*:
assumes *a0*:*env-tran-right* Γ *l rely* $\wedge$ *Sta q rely* $\wedge$ *snd*(*l*!*i*) = *Normal s1* $\wedge$ *s1*$\in$*q*
and
    a1:*Suc i* < *length l* $\wedge$ $\Gamma\vdash_c$(*l*!*i*) $\rightarrow_e$ (*l*!(*Suc i*))
shows $\exists$ *s1 s2*. *snd*(*l*!*i*) = *Normal s1* $\wedge$ *snd*(*l*!(*Suc i*)) = *Normal s2* $\wedge$ *s2*$\in$*q*
using *a0 a1* **unfolding** *env-tran-right-def Sta-def* **by** *fastforce*

lemma *no-env-tran-not-normal*:
assumes *a0*:*env-tran-right* Γ *l rely* $\wedge$ *Sta q rely* $\wedge$ *snd*(*l*!*i*) = *Normal s1* $\wedge$ *s1*$\in$*q*
and
    a1:*Suc i* < *length l* $\wedge$ $\Gamma\vdash_c$(*l*!*i*) $\rightarrow_e$ (*l*!(*Suc i*)) **and**
     a2:($\forall$ *s1*. $\neg$ (*snd*(*l*!*i*) = *Normal s1*)) $\vee$ ($\forall$ *s2*. $\neg$ (*snd* (*l*!*Suc i*) = *Normal s2*))
shows *P*
using *a0 a1 a2* **unfolding** *env-tran-right-def Sta-def* **by** *fastforce*

definition *assum* ::
  (*'s set* $\times$ (*'s*,*'f*) *tran set*) $\Rightarrow$ ((*'s*,*'p*,*'f*,*'e*) *confs*) *set* **where**
  assum $\equiv$ λ(*pre*, *rely*).
      {*c*. *snd*((*snd c*)!*0*) $\in$ *Normal* ' *pre* $\wedge$
        ($\forall$ *i*. *Suc i*<*length* (*snd c*) $\longrightarrow$
        (*fst c*)$\vdash_c$((*snd c*)!*i*) $\rightarrow_e$ ((*snd c*)!(*Suc i*)) $\longrightarrow$
          (*snd*((*snd c*)!*i*), *snd*((*snd c*)!(*Suc i*))) $\in$ *rely*)}

definition *assum1* ::
  (*'s set* $\times$ (*'s*,*'f*) *tran set*) $\Rightarrow$
   'f set $\Rightarrow$
    ((*'s*,*'p*,*'f*,*'e*) *confs*) *set* **where**
  assum1 $\equiv$ λ(*pre*, *rely*) *F*.
      {(Γ,*comp*). *snd*(*comp*!*0*) $\in$ *Normal* ' *pre* $\wedge$
        ($\forall$ *i*. *Suc i*<*length comp* $\longrightarrow$
        $\Gamma\vdash_c$(*comp*!*i*) $\rightarrow_e$ (*comp*!(*Suc i*)) $\longrightarrow$
          (*snd*(*comp*!*i*), *snd*(*comp*!(*Suc i*))) $\in$ *rely*)}

lemma *assum-R-R′*:
  (Γ, *l*) ∈ *assum*(*p*, *R*) ⟹
    snd(*l*!*0*) ∈ *Normal* ‘ *p′* ⟹
    R ⊆ *R′* ⟹
   (Γ, *l*) ∈ *assum*(*p′*, *R′*)
proof −
assume *a0*:(Γ, *l*) ∈ *assum*(*p*, *R*) **and**
       a1:*snd*(*l*!*0*) ∈ *Normal* ‘ *p′* **and**
       a2: *R* ⊆ *R′*
  then have *env-tran-right* Γ *l R*
    unfolding *assum-def* **using** *env-tran-right-def*
    by *force*
  then have *env-tran-right* Γ *l R′*
    using *a2 env-tran-R-R′* **by** *blast*
  thus *?thesis* **using** *a1* **unfolding** *assum-def* **unfolding** *env-tran-right-def*
    by *fastforce*
qed

lemma *same-prog-p*:
  (Γ,(*P*,*s*)#(*P*,*t*)#*l*)∈*cptn* ⟹
   (Γ,(*P*,*s*)#(*P*,*t*)#*l*) ∈ *assum* (*p*,*R*) ⟹
   Sta p R ⟹
   ∃ *t1*. *t*=*Normal t1* ∧ *t1* ∈ *p*

proof −
assume *a0*: (Γ,(*P*,*s*)#(*P*,*t*)#*l*)∈*cptn* **and**
        a1: (Γ,(*P*,*s*)#(*P*,*t*)#*l*) ∈ *assum* (*p*,*R*) **and**
        a2: *Sta p R*
  then have *Suc 0* < *length* ((*P*,*s*)#(*P*,*t*)#*l*)
    by *fastforce*
  then have $\Gamma\vdash_c$(((*P*,*s*)#(*P*,*t*)#*l*)!*0*) $\rightarrow_{ce}$ (((*P*,*s*)#(*P*,*t*)#*l*)!(*Suc 0*))
    using *a0 cptn-stepc-rtran* **by** *fastforce*
  then have *step-ce*:$\Gamma\vdash_c$(((*P*,*s*)#(*P*,*t*)#*l*)!*0*) $\rightarrow_e$ (((*P*,*s*)#(*P*,*t*)#*l*)!(*Suc 0*)) ∨
           $\Gamma\vdash_c$(((*P*,*s*)#(*P*,*t*)#*l*)!*0*) → (((*P*,*s*)#(*P*,*t*)#*l*)!(*Suc 0*))
    using *step-ce-elim-cases* **by** *blast*
  then obtain *s1* **where** *s*:*s*=*Normal s1* ∧ *s1* ∈ *p*
    using *a1* **unfolding** *assum-def*
    by *fastforce*
  have ∃ *t1*. *t*=*Normal t1* ∧ *t1* ∈ *p*
  using *step-ce*
  proof
    {**assume** *step-e*:$\Gamma\vdash_c$ ((*P*, *s*) # (*P*, *t*) # *l*) ! *0* $\rightarrow_e$
        ((*P*, *s*) # (*P*, *t*) # *l*) ! *Suc 0*
     have *?thesis*
     using *a2 a1 s* **unfolding** *Sta-def assum-def*
     proof −

```
      have (Suc 0 < length ((P, s) # (P, t) # l))
        by fastforce
      then have assm:(s, t) ∈ R
        using s a1 step-e
        unfolding assum-def  by fastforce
      then obtain t1 s2 where s-t:s= Normal s2 ∧ t = Normal t1
        using a2 s unfolding Sta-def by fastforce
      then have R:(s,t)∈R
        using assm  unfolding Satis-def by fastforce
      then have s2=s1 using s s-t by fastforce
      then have t1∈p
        using a2 s s-t R unfolding Sta-def Norm-def by blast
      thus ?thesis using s-t by blast
     qed thus ?thesis by auto
    }
    next
    {
     assume step:Γ⊢c ((P, s) # (P, t) # l) ! 0 →
         ((P, s) # (P, t) # l) ! Suc 0
     then have P≠P ∨ s=t
     proof −
       have Γ⊢c (P, s) → (P, t)
         using local.step by force
       then show ?thesis
         using step-change-p-or-eq-s by blast
     qed
     then show ?thesis using s by fastforce
    }
  qed thus  ?thesis by auto
qed

lemma tl-of-assum-in-assum:
  (Γ,(P,s)#(P,t)#l)∈cptn ⟹
   (Γ,(P,s)#(P,t)#l) ∈ assum (p,R) ⟹
   Sta p R  ⟹
   (Γ,(P,t)#l) ∈ assum (p,R)

proof −
  assume a0: (Γ,(P,s)#(P,t)#l)∈cptn and
         a1: (Γ,(P,s)#(P,t)#l) ∈ assum (p,R) and
         a2: Sta p R

  then obtain t1 where t1:t=Normal t1 ∧ t1 ∈p
   using same-prog-p by blast
  then have env-tran-right Γ ((P,s)#(P,t)#l) R
    using env-tran-right-def a1 unfolding assum-def
    by force
  then have env-tran-right Γ ((P,t)#l) R
    using env-tran-tail by auto
```

thus *?thesis* **using** *t1* **unfolding** *assum-def env-tran-right-def* **by** *auto*
qed

lemma *tl-of-assum-in-assum1*:
$(\Gamma,(P,s)\#(Q,t)\#l) \in cptn \Longrightarrow$
$(\Gamma,(P,s)\#(Q,t)\#l) \in assum\ (p,R) \Longrightarrow$
$t \in Normal\ `\ q \Longrightarrow$
$(\Gamma,(Q,t)\#l) \in assum\ (q,R)$

proof −
assume *a0*: $(\Gamma,(P,s)\#(Q,t)\#l) \in cptn$ **and**
a1: $(\Gamma,(P,s)\#(Q,t)\#l) \in assum\ (p,R)$ **and**
a2: $t \in Normal\ `\ q$
then have *env-tran-right* Γ $((P,s)\#(Q,t)\#l)$ R
using *env-tran-right-def a1* **unfolding** *assum-def*
by *force*
then have *env-tran-right* Γ $((Q,t)\#l)$ R
using *env-tran-tail* **by** *auto*
thus *?thesis* **using** *a2* **unfolding** *assum-def env-tran-right-def* **by** *auto*
qed

lemma *sub-assum*:
assumes *a0*: $(\Gamma,(x\#l0)@l1) \in assum\ (p,R)$
shows $(\Gamma,x\#l0) \in assum\ (p,R)$
proof −
{**have** *p0*:*snd* $x \in Normal\ `\ p$
using *a0* **unfolding** *assum-def* **by** *force*
then have *env-tran-right* Γ $((x\#l0)@l1)$ R
using *a0* **unfolding** *assum-def*
by (*auto simp add*: *env-tran-right-def*)
then have *env*:*env-tran-right* Γ $(x\#l0)$ R
using *env-tran-subr* **by** *blast*
also have $snd\ ((x\#l0)!0) \in Normal\ `\ p$
using *p0* **by** *fastforce*
ultimately have $snd\ ((x\#l0)!0) \in Normal\ `\ p\ \wedge$
$(\forall i.\ Suc\ i<length\ (x\#l0) \longrightarrow$
$\Gamma\vdash_c((x\#l0)!i) \rightarrow_e ((x\#l0)!(Suc\ i)) \longrightarrow$
$(snd((x\#l0)!i),\ snd((x\#l0)!(Suc\ i))) \in R)$
unfolding *env-tran-right-def* **by** *auto*
}
then show *?thesis* **unfolding** *assum-def* **by** *auto*
qed

lemma *sub-assum-r*:
assumes *a0*: $(\Gamma,l0@x1\#l1) \in assum\ (p,R)$ **and**
a1: $(snd\ x1) \in Normal\ `\ q$
shows $(\Gamma,x1\#l1) \in assum\ (q,R)$
proof −
have *env-tran-right* Γ $(l0@x1\#l1)$ R

using *a0* **unfolding** *assum-def env-tran-right-def*
by *fastforce*
then have *env-tran-right* Γ *(x1#l1) R*
using *env-tran-subl* **by** *auto*
thus *?thesis* **using** *a1* **unfolding** *assum-def env-tran-right-def* **by** *fastforce*
qed

definition *comm* ::
$(('s,'f)\ tran)\ set\ \times$
$('s\ set\ \times\ 's\ set) \Rightarrow$
$'f\ set \Rightarrow$
$(('s,'p,'f,'e)\ confs)\ set$ **where**
$comm \equiv \lambda(guar,\ (q,a))\ F.$
$\{c.\ snd\ (last\ (snd\ c)) \notin Fault\ `\ F\ \longrightarrow$
$(\forall i.$
$Suc\ i{<}length\ (snd\ c) \longrightarrow$
$(fst\ c)\vdash_c((snd\ c)!i)\ \rightarrow ((snd\ c)!(Suc\ i)) \longrightarrow$
$(snd((snd\ c)!i),\ snd((snd\ c)!(Suc\ i))) \in guar) \wedge$
$(final\ (last\ (snd\ c))\ \longrightarrow$
$((fst\ (last\ (snd\ c)) = Skip\ \wedge$
$snd\ (last\ (snd\ c)) \in Normal\ `\ q)) \vee$
$(fst\ (last\ (snd\ c)) = Throw\ \wedge$
$snd\ (last\ (snd\ c)) \in Normal\ `\ a))\}$

definition *comm1* ::
$(('s,'f)\ tran)\ set\ \times$
$('s\ set\ \times\ 's\ set) \Rightarrow$
$'f\ set \Rightarrow$
$(('s,'p,'f,'e)\ confs)\ set$ **where**
$comm1 \equiv \lambda(guar,\ (q,a))\ F.$
$\{(\Gamma,comp).\ snd\ (last\ comp) \notin Fault\ `\ F\ \longrightarrow$
$(\forall i.$
$Suc\ i{<}length\ comp \longrightarrow$
$\Gamma\vdash_c(comp!i)\ \rightarrow (comp!(Suc\ i)) \longrightarrow$
$(snd(comp!i),\ snd(comp!(Suc\ i))) \in guar) \wedge$
$(final\ (last\ comp)\ \longrightarrow$
$((fst\ (last\ comp) = Skip\ \wedge$
$snd\ (last\ comp) \in Normal\ `\ q)) \vee$
$(fst\ (last\ comp) = Throw\ \wedge$
$snd\ (last\ comp) \in Normal\ `\ a))\}$

lemma *comm-dest*:
$(\Gamma,\ l)\in\ comm\ (G,(q,a))\ F \Longrightarrow$
$snd\ (last\ l) \notin Fault\ `\ F \Longrightarrow$
$(\forall i.\ Suc\ i{<}length\ l \longrightarrow$
$\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in\ G)$
unfolding *comm-def*
apply *clarify*

apply (*drule mp*)
apply *fastforce*
apply (*erule conjE*)
apply (*erule allE*)
by *auto*

lemma *comm-dest1*:
$(\Gamma,\ l) \in$ *comm* $(G,(q,a))$ $F \Longrightarrow$
snd (*last* l) $\notin$ *Fault* ‘ $F \Longrightarrow$
Suc i<*length* $l \Longrightarrow$
$\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \Longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G$
unfolding *comm-def*
apply *clarify*
apply (*drule mp*)
apply *fastforce*
apply (*erule conjE*)
apply (*erule allE*)
by *auto*

lemma *comm-dest2*:
  assumes *a0*: $(\Gamma,\ l) \in$ *comm* $(G,(q,a))$ F **and**
        a1: *final* (*last* l) **and**
        a2: *snd* (*last* l) $\notin$ *Fault* ‘ F
  shows ((*fst* (*last* l) = *Skip* $\wedge$
        snd (*last* l) $\in$ *Normal* ‘ q)) $\vee$
        (*fst* (*last* l) = *Throw* $\wedge$
        snd (*last* l) $\in$ *Normal* ‘ a)
proof −
  show *?thesis* **using** *a0 a1 a2* **unfolding** *comm-def* **by** *auto*
qed

lemma *comm-des3*:
  assumes *a0*: $(\Gamma,\ l) \in$ *comm* $(G,(q,a))$ F **and**
        a1: *snd* (*last* l) $\notin$ *Fault* ‘ F
 shows *final* (*last* l) $\longrightarrow$ ((*fst* (*last* l) = *Skip* $\wedge$
        snd (*last* l) $\in$ *Normal* ‘ q)) $\vee$
        (*fst* (*last* l) = *Throw* $\wedge$
        snd (*last* l) $\in$ *Normal* ‘ a)
using *a0 a1* **unfolding** *comm-def* **by** *auto*

lemma *commI*:
  assumes *a0*:*snd* (*last* l) $\notin$ *Fault* ‘ $F \Longrightarrow$
      ($\forall\, i$.
        Suc i<*length* $l \longrightarrow$
        $\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
          $(snd(l!i),\ snd(l!(Suc\ i))) \in G) \wedge$
        (*final* (*last* l) $\longrightarrow$
          ((*fst* (*last* l) = *Skip* $\wedge$

$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ a))$
shows $(\Gamma,l)\in comm\ (G,\ (q,a))\ F$
using *a0* **unfolding** *comm-def*
apply *clarify*
by *simp*

lemma *comm-conseq*:
$(\Gamma,l) \in comm(G',\ (q',a'))\ F \Longrightarrow$
$G' \subseteq G \wedge$
$q' \subseteq q \wedge$
$a' \subseteq a \Longrightarrow$
$(\Gamma,l) \in comm\ (G,(q,a))\ F$
proof –
assume *a0*:$(\Gamma,l) \in comm(G',\ (q',a'))\ F$ **and**
a1: $G' \subseteq G\ \wedge$
$q' \subseteq q \wedge$
$a' \subseteq a$
{
assume *a*:$snd\ (last\ l) \notin Fault\ `\ F$
have l:$(\forall\ i.$
$Suc\ i<length\ l \longrightarrow$
$\Gamma\vdash_c(l!i)\ \ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof –
{fix $i\ ns\ ns'$
assume *a00*:$Suc\ i<length\ l$ **and**
a11:$\Gamma\vdash_c(l!i)\ \ \rightarrow (l!(Suc\ i))$
have $(snd(l!i),\ snd(l!(Suc\ i))) \in\ G$
proof –
have $(snd(l!i),\ snd(l!(Suc\ i))) \in\ G'$
using *comm-dest1* [*OF a0 a a00 a11*] **by** *auto*
thus *?thesis* **using** *a1* **unfolding** *Satis-def sep-conj-def* **by** *fastforce*
qed
} thus *?thesis* **by** *auto*
qed
have $(final\ (last\ l)\ \ \longrightarrow$
$((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ a))$
proof –
{assume *a33*:$final\ (last\ l)$
then have $((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q')) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ a')$
using *comm-dest2*[*OF a0 a33 a*] **by** *auto*

then have ((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ‘ *a*)
using *a1* **by** *fastforce*
} **thus** *?thesis* **by** *auto*
qed
note *res1* = *conjI*[*OF l this*]
} **thus** *?thesis* **unfolding** *comm-def* **by** *simp*
qed

lemma *no-comp-tran-no-final-comm*:
assumes *a0*:$\forall$ *i*<*length l*. ¬ *final* (*l*!*i*) **and**
a1:$\forall$ *i*<*length l*. *fst* (*l*!*i*) = *C* **and** *a2*:*length l*>*0*
shows (Γ,*l*)∈*comm*(*G*, (*q*,*a*)) *F*
proof−
have *n-comp*:$\forall$ *i*. *Suc i* < *length l* ⟶ ¬ ($\Gamma\vdash_c$ (*l*!*i*) → (*l*!*Suc i*)) **using** *a1*
by (*metis Suc-lessD mod-env-not-component prod.collapse*)
{assume *a00*:*snd* (*last l*) ∉ *Fault* ‘ *F*
{fix *i*
assume *Suc i*< *length*(*l*) **and**
$\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*))
then have *False* **using** *n-comp* **by** *auto*
}
moreover {
assume *final* (*last l*)
then have *False* **using** *a0 a2*
using *last-conv-nth* **by** *force*
}
ultimately have *?thesis* **unfolding** *comm-def* **using** *a00* **by** *auto*
} **thus** *?thesis* **unfolding** *comm-def* **by** *auto*
qed

definition *com-validity* ::
(*'s*,*'p*,*'f*,*'e*) *body* ⇒ *'f set* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ⇒
's set ⇒ ((*'s*,*'f*) *tran*) *set* ⇒ ((*'s*,*'f*) *tran*) *set* ⇒
's set ⇒ *'s set* ⇒ *bool*
(- $\models_{'/\text{-}}$/ - *sat* [-,-, -, -,-] [*61*,*60*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
$\Gamma \models_{/F}$ *Pr sat* [*p*, *R*, *G*, *q*,*a*] ≡
$\forall$ *s*. *cp* Γ *Pr s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*

definition *com-cvalidity*::
(*'s*,*'p*,*'f*,*'e*) *body* ⇒
(*'s*,*'p*,*'f*,*'e*) *sextuple set* ⇒
'f set ⇒
(*'s*,*'p*,*'f*,*'e*) *com* ⇒
's set ⇒
((*'s*,*'f*) *tran*) *set* ⇒
((*'s*,*'f*) *tran*) *set* ⇒

$'s$ *set* ⇒
$'s$ *set* ⇒
bool
(-,- $\models_{'/\_}$/ - *sat* [-,-, -, -,-] [*61*,*60*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
$\Gamma,\Theta \models_{/F}$ *Pr sat* [p, R, G, q,a] ≡
($\forall$ (c,p,R,G,q,a)∈ Θ. $\Gamma \models_{/F}$ (*Call c*) *sat* [p, R, G, q,a]) ⟶
$\Gamma \models_{/F}$ *Pr sat* [p, R, G, q,a]

definition *com-validityn* ::
($'s$,$'p$,$'f$,$'e$) *body* ⇒ *nat* ⇒ $'f$ *set* ⇒ ($'s$,$'p$,$'f$,$'e$) *com* ⇒
$'s$ *set* ⇒ (($'s$,$'f$) *tran*) *set* ⇒ (($'s$,$'f$) *tran*) *set* ⇒
$'s$ *set* ⇒ $'s$ *set* ⇒ *bool*
(- $\models$-$_{'/\_}$/ - *sat* [-,-, -, -,-] [*61*,*0*,*60*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
$\Gamma \models n_{/F}$ *Pr sat* [p, R, G, q,a] ≡
$\forall$ s. *cpn n* Γ *Pr s* ∩ *assum*(p, R) ⊆ *comm*(G, (q,a)) F

definition *com-cvalidityn*::
($'s$,$'p$,$'f$,$'e$) *body* ⇒
($'s$,$'p$,$'f$,$'e$) *sextuple set* ⇒ *nat* ⇒
$'f$ *set* ⇒
($'s$,$'p$,$'f$,$'e$) *com* ⇒
$'s$ *set* ⇒
(($'s$,$'f$) *tran*) *set* ⇒
(($'s$,$'f$) *tran*) *set* ⇒
$'s$ *set* ⇒
$'s$ *set* ⇒
bool
(-,- $\models$-$_{'/\_}$/ - *sat* [-,-, -, -,-] [*61*,*60*,*0*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
$\Gamma,\Theta \models n_{/F}$ *Pr sat* [p, R, G, q,a] ≡
($\forall$ (c,p,R,G,q,a)∈ Θ. $\Gamma \models n_{/F}$ (*Call c*) *sat* [p, R, G, q,a]) ⟶
$\Gamma \models n_{/F}$ *Pr sat* [p, R, G, q,a]

lemma *com-valid-iff-nvalid*:($\Gamma \models_{/F}$ *Pr sat* [p, R, G, q,a]) = ($\forall$ n. $\Gamma \models n_{/F}$ *Pr sat* [p, R, G, q,a])
apply (*simp only*: *com-validity-def com-validityn-def cp-def cpn-def cptn-eq-cptn-mod-nest*)
by *fast*

lemma *com-cnvalid-to-cvalid*: ($\forall$ n. ($\Gamma,\Theta \models n_{/F}$ *Pr sat* [p, R, G, q,a])) ⟹ ($\Gamma,\Theta \models_{/F}$ *Pr sat* [p, R, G, q,a])
apply (*unfold com-cvalidityn-def com-cvalidity-def com-valid-iff-nvalid* [*THEN eq-reflection*])
by *fast*

lemma *etran-in-comm*:
(Γ,(P, t) # *xs*) ∈ *comm*(G, (q,a)) F ⟹

```
  ¬ (Γ⊢c((P,s)) → ((P,t))) ⟹
  (Γ,(P, s) # (P, t) # xs) ∈ cptn ⟹
 (Γ,(P, s) # (P, t) # xs) ∈ comm(G, (q,a)) F
proof -
  assume a1:(Γ,(P, t) # xs) ∈ comm(G, (q,a)) F and
         a2:¬ Γ⊢c((P,s)) → ((P,t)) and
         a3:(Γ,(P, s) # (P, t) # xs) ∈ cptn
  show ?thesis using comm-def a1 a2 a3
  proof -
    {
    let ?l1 = (P, t) # xs
    let ?l = (P, s) # ?l1
    assume a00:snd (last ?l) ∉ Fault ‘ F
    have concl:(∀ i ns ns'. Suc i<length ?l ⟶
              Γ⊢c(?l!i) → (?l!(Suc i)) ⟶
                (snd(?l!i), snd(?l!(Suc i))) ∈ G)
    proof -
      {fix i ns ns'
       assume a11:Suc i < length ?l and
              a12:Γ⊢c (?l ! i) → ( ?l ! Suc i)
       have p1:(∀ i ns ns'. Suc i<length ?l1 ⟶
              Γ⊢c(?l1!i) → (?l1!(Suc i)) ⟶
              (snd(?l1!i), snd(?l1!(Suc i))) ∈ G)
       using a1 a3 a00 unfolding comm-def by auto
       have (snd (?l ! i), snd (?l ! Suc i)) ∈ G
       proof (cases i)
         case 0
         have Γ⊢c (P, s) → (P, t) using a12 0 by auto
         thus ?thesis using a2 by auto
       next
         case (Suc n) thus ?thesis
         proof -
           have f1: Γ⊢c ((P, t) # xs) ! n → ((P, t) # xs) ! Suc n
             using Suc a12 by fastforce
           have f2: Suc n < length ((P, t) # xs)
             using Suc a11 by fastforce
           have snd (last ((P, t) # xs)) ∉ Fault ‘ F
               by (metis (no-types) a00 last.simps list.distinct(1))
           hence (snd (((P, t) # xs) ! n), snd (((P, t) # xs) ! Suc n)) ∈ G
             using f2 f1 a1 comm-dest1 by blast
           thus ?thesis
             by (simp add: Suc)
         qed
       qed
      } thus ?thesis by auto
    qed
    have concr:(final (last ?l) ⟶
                 ((fst (last ?l) = Skip ∧
                   snd (last ?l) ∈ Normal ‘ q)) ∨
```

(*fst* (*last ?l*) = *Throw* ∧
snd (*last ?l*) ∈ *Normal* ' *a*))
using *a1 a00* **unfolding** *comm-def* **by** *auto*
note *res1=conjI*[*OF concl concr*] **}**
thus *?thesis* **unfolding** *comm-def* **by** *auto* **qed**
qed

lemma *ctran-in-comm*:
(*Normal s*,*Normal s*) ∈ *G* ⟹
(Γ,(*Q*, *Normal s*) # *xs*) ∈ *comm*(*G*, (*q*,*a*)) *F* ⟹
(Γ,(*P*, *Normal s*) # (*Q*, *Normal s*) # *xs*) ∈ *comm*(*G*, (*q*,*a*)) *F*
proof −
assume *a1*:(*Normal s*,*Normal s*) ∈ *G* **and**
a2:(Γ,(*Q*, *Normal s*) # *xs*) ∈ *comm*(*G*, (*q*,*a*)) *F*
show *?thesis* **using** *comm-def a1 a2*
proof −
{
let *?l1* = (*Q*, *Normal s*) # *xs*
let *?l* = (*P*, *Normal s*) # *?l1*
assume *a00*:*snd* (*last ?l*) ∉ *Fault* ' *F*
have *concl*:(∀ *i*. *Suc i*<*length ?l* ⟶
Γ⊢$_c$(*?l*!*i*) → (*?l*!(*Suc i*)) ⟶
(*snd*(*?l*!*i*), *snd*(*?l*!(*Suc i*))) ∈ *G*)
proof −
{fix *i ns ns'*
assume *a11*:*Suc i* < *length ?l* **and**
a12:Γ⊢$_c$ (*?l* ! *i*) → (*?l* ! *Suc i*)
have *p1*:(∀ *i*. *Suc i*<*length ?l1* ⟶
Γ⊢$_c$(*?l1*!*i*) → (*?l1*!(*Suc i*)) ⟶
(*snd*(*?l1*!*i*), *snd*(*?l1*!(*Suc i*))) ∈ *G*)
using *a2 a00* **unfolding** *comm-def* **by** *auto*
have (*snd* (*?l* ! *i*), *snd* (*?l* ! *Suc i*)) ∈ *G*
proof (*cases i*)
case *0*
then have *snd* (((*P*, *Normal s*) # (*Q*, *Normal s*) # *xs*) ! *i*) = *Normal s*
∧
snd (((*P*, *Normal s*) # (*Q*, *Normal s*) # *xs*) ! (*Suc i*)) = *Normal*
s
by *fastforce*
also have (*Normal s*, *Normal s*) ∈ *G*
using *Satis-def a1* **by** *blast*
ultimately show *?thesis* **using** *a1 Satis-def* **by** *auto*
next
case (*Suc n*) **thus** *?thesis* **using** *p1 a2 a11 a12*
proof −
have *f1*: Γ⊢$_c$ ((*Q*, *Normal s*) # *xs*) ! *n* → ((*Q*, *Normal s*) # *xs*) ! *Suc n*
using *Suc a12* **by** *fastforce*
have *f2*: *Suc n* < *length* ((*Q*, *Normal s*) # *xs*)
using *Suc a11* **by** *fastforce*

thus *?thesis* **using** *Suc f1 nth-Cons-Suc p1* **by** *auto*
qed
qed
} **thus** *?thesis* **by** *auto*
qed
have *concr*:(*final* (*last ?l*) ⟶
snd (*last ?l*) ∉ *Fault* ‘ *F* ⟶
((*fst* (*last ?l*) = *Skip* ∧
snd (*last ?l*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last ?l*) = *Throw* ∧
snd (*last ?l*) ∈ *Normal* ‘ *a*))
using *a2* **unfolding** *comm-def* **by** *auto*
note *res*=*conjI*[*OF concl concr*]**}**
thus *?thesis* **unfolding** *comm-def* **by** *auto* **qed**
qed

lemma *not-final-in-comm*:
(Γ,(*Q*, *Normal s*) # *xs*) ∈ *comm*(*G*, (*q*,*a*)) *F* ⟹
¬ *final* (*last* ((*Q*, *Normal s*) # *xs*)) ⟹
(Γ,(*Q*, *Normal s*) # *xs*) ∈ *comm*(*G*, (*q'*,*a'*)) *F*
unfolding *comm-def* **by** *force*

lemma *comm-union*:
assumes
a0: (Γ,*xs*) ∈ *comm*(*G*, (*q*,*a*)) *F* **and**
a1: (Γ,*ys*) ∈ *comm*(*G*, (*q'*,*a'*)) *F* **and**
a2: *xs*≠[] ∧ *ys*≠[] **and**
a3: (*snd* (*last xs*),*snd* (*ys*!*0*)) ∈ *G* **and**
a4: (Γ,*xs*@*ys*) ∈ *cptn*
shows (Γ,*xs*@*ys*) ∈ *comm*(*G*, (*q'*,*a'*)) *F*
proof −
{
let *?l*=*xs*@*ys*
assume *a00*:*snd* (*last* (*xs*@*ys*)) ∉ *Fault* ‘ *F*
have *last-ys*:*last* (*xs*@*ys*) = *last ys* **using** *a2* **by** *fastforce*
have *concl*:(∀ *i*. *Suc i*<*length ?l* ⟶
Γ⊢$_c$(*?l*!*i*) → (*?l*!(*Suc i*)) ⟶
(*snd*(*?l*!*i*), *snd*(*?l*!(*Suc i*))) ∈ *G*)
proof −
{fix *i ns ns'*
assume *a11*:*Suc i* < *length ?l* **and**
a12:Γ⊢$_c$ (*?l* ! *i*) → (*?l* ! *Suc i*)
have *all-ys*:∀ *i*≥*length xs*. (*xs*@*ys*)!*i* = *ys*!(*i*−(*length xs*))
by (*simp add*: *nth-append*)
have *all-xs*:∀ *i*<*length xs*. (*xs*@*ys*)!*i* = *xs*!*i*
by (*simp add*: *nth-append*)
have (*snd*(*?l*!*i*), *snd*(*?l*!(*Suc i*))) ∈ *G*
proof (*cases Suc i*>*length xs*)
case *True*

have *Suc* $(i - (length\ xs)) <$ *length ys* **using** *a11 True* **by** *fastforce*
moreover have $\Gamma\vdash_c (ys\ !\ (i-(length\ xs))) \rightarrow (\ ys\ !\ ((Suc\ i)-(length\ xs)))$
using *a12 all-ys True* **by** *fastforce*
moreover have *snd* (*last ys*) $\notin$ *Fault* ' *F* **using** *last-ys a00* **by** *fastforce*
ultimately have $(snd(ys!(i-(length\ xs))), snd(ys!Suc\ (i-(length\ xs)))) \in G$
using *a1 comm-dest1*[*of* Γ *ys G* q' a' *F i*$-$*length xs*] *True Suc-diff-le* **by** *fastforce*
thus *?thesis* **using** *True all-ys Suc-diff-le* **by** *fastforce*
next
case *False* **note** *F1=this* **thus** *?thesis*
proof (*cases Suc i* $<$ *length xs*)
case *True*
then have *snd* ((*xs*@*ys*)!(*length xs* $-$*1*)) $\notin$ *Fault* ' *F*
using *a00 a2 a4*
by (*simp add*: *last-not-F*)
then have *snd* (*last xs*) $\notin$ *Fault* ' *F* **using** *all-xs a2* **by** (*simp add*: *last-conv-nth*)
moreover have $\Gamma\vdash_c (xs\ !\ i) \rightarrow (\ xs\ !\ Suc\ i)$
using *True all-xs a12* **by** *fastforce*
ultimately have(*snd*(*xs*!*i*), *snd*(*xs*!(*Suc i*))) $\in$ *G*
using *a0 comm-dest1*[*of* Γ *xs G q a F i*] *True* **by** *fastforce*
thus *?thesis* **using** *True all-xs* **by** *fastforce*
next
case *False*
then have *suc-i*:*Suc i* = *length xs* **using** *F1* **by** *fastforce*
then have *i*:*i*=*length xs* $-$*1* **using** *a2* **by** *fastforce*
then show *?thesis* **using** *a3*
by (*simp add*: *a2 all-xs all-ys last-conv-nth*)
qed
qed
} **thus** *?thesis* **by** *auto*
qed
have *concr*:(*final* (*last ?l*) $\longrightarrow$
((*fst* (*last ?l*) = *Skip* $\wedge$
snd (*last ?l*) $\in$ *Normal* ' q')) $\vee$
(*fst* (*last ?l*) = *Throw* $\wedge$
snd (*last ?l*) $\in$ *Normal* ' a'))
using *a1 last-ys a00 a2 comm-des3* **by** *fastforce*
note *res=conjI*[*OF concl concr*]**}**
thus *?thesis* **unfolding** *comm-def* **by** *auto*
qed

lemma *cpn-rule1*:($\forall$ *s*. *cpn n* Γ *P s* $\cap$ *assum*(*p*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*) $\Longrightarrow$
($\forall$ *s l*. (Γ,*l*)$\in$*cpn n* Γ *P s* $\wedge$ (Γ,*l*)$\in$ *assum* (*p*, *R*) $\longrightarrow$ (Γ,*l*) $\in$ *comm*(*G*, (*q*,*a*)) *F*)
proof$-$

assume *a0*:∀ *s*. *cpn n* Γ *P s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*
{**fix** *s l*
assume *a00*:(Γ,*l*)∈*cpn n* Γ *P s* ∧ (Γ,*l*)∈ *assum* (*p*, *R*)
then have *cpn n* Γ *P s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F* **using** *a0* **by** *auto*
then have (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F* **using** *a00* **unfolding** *cpn-def assum-def comm-def*
by *blast*
} **then show** *?thesis* **by** *auto*
qed

lemma *cpn-rule2*:(∀ *s l*. (Γ,*l*)∈*cpn n* Γ *P s* ∧ (Γ,*l*)∈ *assum* (*p*, *R*) ⟶ (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F*) ⟹
(∀ *s*. *cpn n* Γ *P s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*)
proof−
assume *a0*:∀ *s l*. (Γ,*l*)∈*cpn n* Γ *P s* ∧ (Γ,*l*)∈ *assum* (*p*, *R*) ⟶ (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F*
{**fix** *s l*
assume *a00*:(Γ,*l*)∈ *cpn n* Γ *P s* ∧ (Γ,*l*)∈*assum*(*p*, *R*)
then have (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F* **using** *a0* **unfolding** *cpn-def assum-def comm-def*
by *blast*
} **then show** *?thesis* **unfolding** *cpn-def* **by** *fastforce*
qed

lemma *cpn-rule*:(∀ *s l*. (Γ,*l*)∈*cpn n* Γ *P s* ∧ (Γ,*l*)∈ *assum* (*p*, *R*) ⟶ (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F*) =
(∀ *s*. *cpn n* Γ *P s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*)
using *cpn-rule1 cpn-rule2*
by *metis*

lemma *split-list-i*:*i*<*length l* ⟹
∃ *l1 l2*. *l* = *l1*@(*l*!*i*#*l2*)
proof(*induct l arbitrary*: *i*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a l*)
then show *?case*
using *id-take-nth-drop* **by** *blast*
qed

lemma *sub-assum1*:
assumes *a0*: (Γ,*l0*@*l1*) ∈ *assum* (*p*,*R*) **and** *a1*:*l0*≠[]
shows (Γ,*l0*) ∈ *assum* (*p*,*R*)
by (*metis a0 a1 append-self-conv2 id-take-nth-drop length-greater-0-conv sub-assum take-0*)

11.3 Validity for Parallel Programs.

definition *All-End* :: $('s,'p,'f,'e)$ *par-config* $\Rightarrow$ *bool* **where**
All-End xs $\equiv$ *fst xs* $\neq$[] $\wedge$ ($\forall$ *i*<*length* (*fst xs*). *final* ((*fst xs*)!*i*,*snd xs*))

definition *par-assum* ::
$('s$ *set* $\times$
$(('s,'f)$ *tran*) *set*) $\Rightarrow$
$(('s,'p,'f,'e)$ *par-confs*) *set* **where**
par-assum $\equiv$
λ(*pre*, *rely*). {*c*.
snd((*snd c*)!*0*) $\in$ *Normal* ' *pre* $\wedge$ ($\forall$ *i*. *Suc i*<*length* (*snd c*) $\longrightarrow$
(*fst c*)$\vdash_p$((*snd c*)!*i*) $\rightarrow_e$ ((*snd c*)!(*Suc i*)) $\longrightarrow$
(*snd*((*snd c*)!*i*), *snd*((*snd c*)!(*Suc i*))) $\in$ *rely*)}

definition *par-comm* ::
$((('s,'f)$ *tran*) *set* $\times$
$('s$ *set* $\times$ $'s$ *set*)) $\Rightarrow$
$'f$ *set* $\Rightarrow$
$(('s,'p,'f,'e)$ *par-confs*) *set* **where**
par-comm $\equiv$
λ(*guar*, (*q*,*a*)) *F*.
{*c*. *snd* (*last* (*snd c*)) $\notin$ *Fault* ' *F* $\longrightarrow$
($\forall$ *i*.
Suc i<*length* (*snd c*) $\longrightarrow$
(*fst c*)$\vdash_p$((*snd c*)!*i*) $\rightarrow$ ((*snd c*)!(*Suc i*)) $\longrightarrow$
(*snd*((*snd c*)!*i*), *snd*((*snd c*)!(*Suc i*))) $\in$ *guar*) $\wedge$
(*All-End* (*last* (*snd c*)) $\longrightarrow$
($\exists$ *j*<*length* (*fst* (*last* (*snd c*))). *fst* (*last* (*snd c*))!*j*=*Throw* $\wedge$
snd (*last* (*snd c*)) $\in$ *Normal* ' *a*) $\vee$
($\forall$ *j*<*length* (*fst* (*last* (*snd c*))). *fst* (*last* (*snd c*))!*j*=*Skip* $\wedge$
snd (*last* (*snd c*)) $\in$ *Normal* ' *q*))}

definition *par-com-validity* ::
$('s,'p,'f,'e)$ *body* $\Rightarrow$
$'f$ *set* $\Rightarrow$
$('s,'p,'f,'e)$ *par-com* $\Rightarrow$
$('s$ *set*) $\Rightarrow$
$((('s,'f)$ *tran*) *set*) $\Rightarrow$
$((('s,'f)$ *tran*) *set*) $\Rightarrow$
$('s$ *set*) $\Rightarrow$
$('s$ *set*) $\Rightarrow$
bool
(- $\models_{'/\text{-}}$/ - *SAT* [-, -, -, -,-] [*61*,*60*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
$\Gamma \models_{/F}$ *Ps SAT* [*pre*, *R*, *G*, *q*,*a*] $\equiv$
$\forall$ *s*. *par-cp* Γ *Ps s* $\cap$ *par-assum*(*pre*, *R*) $\subseteq$ *par-comm*(*G*, (*q*,*a*)) *F*

definition *par-com-cvalidity* ::
$('s,'p,'f,'e)$ *body* $\Rightarrow$

('s,'p,'f,'e) *sextuple set* ⇒
'f *set* ⇒
('s,'p,'f,'e) *par-com* ⇒
('s *set*) ⇒
((('s,'f) *tran*) *set*) ⇒
((('s,'f) *tran*) *set*) ⇒
('s *set*) ⇒
('s *set*) ⇒
bool
(-,- $\models_{'/-}$/ - *SAT* [-, -, -, -,-] [*61*,*60*,*0*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
Γ,Θ $\models_{/F}$ *Ps SAT* [*p*, *R*, *G*, *q*,*a*] ≡
(∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. (Γ $\models_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*])) ⟶
Γ $\models_{/F}$ *Ps SAT* [*p*, *R*, *G*, *q*,*a*]

declare *Un-subset-iff* [*simp del*] *sup.bounded-iff* [*simp del*]

inductive
lrghoare :: [('s,'p,'f,'e) *body*,
('s,'p,'f,'e) *sextuple set*,
'f *set*,
('s,'p,'f,'e) *com*,
('s *set*),
(('s,'f) *tran*) *set*, (('s,'f) *tran*) *set*,
's *set*,
's *set*] ⇒ *bool*
(-,- $\vdash_{'/-}$ - *sat* [-, -, -, -,-] [*61*,*61*,*60*,*60*,*0*,*0*,*0*,*0*] *45*)
where
Skip: ⟦ *Sta q R*; (∀ *s*. (*Normal s*, *Normal s*) ∈ *G*) ⟧ ⟹
Γ,Θ $\vdash_{/F}$ *Skip sat* [*q*, *R*, *G*, *q*,*a*]

|*Spec*: ⟦*Sta p R*;*Sta q R*;
(∀ *s t*. *s*∈*p* ∧ (*s*,*t*)∈*r* ⟶ (*Normal s*,*Normal t*) ∈ *G*);
p ⊆ {*s*. (∀ *t*. (*s*,*t*)∈*r* ⟶ *t* ∈ *q*) ∧ (∃ *t*. (*s*,*t*) ∈ *r*)} ⟧ ⟹
Γ,Θ $\vdash_{/F}$ (*Spec r e*) *sat* [*p*, *R*, *G*, *q*,*a*]

| *Basic*: ⟦ *Sta p R*;*Sta q R*;
(∀ *s t*. *s*∈*p* ∧ (*t*=*f s*) ⟶ (*Normal s*,*Normal t*) ∈ *G*);
p ⊆ {*s*. *f s* ∈ *q*} ⟧ ⟹
Γ,Θ $\vdash_{/F}$ (*Basic f e*) *sat* [*p*, *R*, *G*, *q*,*a*]

| *If*: ⟦*Sta p R*; (∀ *s*. (*Normal s*, *Normal s*) ∈ *G*);
Γ,Θ $\vdash_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *q*,*a*];
Γ,Θ $\vdash_{/F}$ *c2 sat* [*p* ∩ (−*b*), *R*, *G*, *q*,*a*]⟧ ⟹
Γ,Θ $\vdash_{/F}$ (*Cond b c1 c2*) *sat* [*p*, *R*, *G*, *q*,*a*]

| *While*: ⟦ *Sta p R*; *Sta* (*p* ∩ (−*b*)) *R*; *Sta a R*; (∀ *s*. (*Normal s*,*Normal s*) ∈ *G*);
Γ,Θ $\vdash_{/F}$ *c sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*]⟧ ⟹
Γ,Θ $\vdash_{/F}$ (*While b c*) *sat* [*p*, *R*, *G*, *p* ∩ (−*b*),*a*]

| *Seq*: $[\![$*Sta a R*; *Sta p R*; $(\forall s.\ (Normal\ s, Normal\ s) \in G)$;
$\Gamma,\Theta \vdash_{/F}$ *c1 sat* $[p,\ R,\ G,\ q,a]$; $\Gamma,\Theta \vdash_{/F}$ *c2 sat* $[q,\ R,\ G,\ r,a]]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Seq c1 c2) sat* $[p,\ R,\ G,\ r,a]$

| *Await*: $[\![$ *Sta p R*; *Sta q R*; *Sta a R*;
$\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
$(p \cap b \cap \{V\})\ c$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q)$,
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a)]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Await b c e) sat* $[p,\ R,\ G,\ q,a]$

| *Guard*: $[\![$*Sta* $(p \cap g)$ *R*; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$;
$\Gamma,\Theta \vdash_{/F}$ *c sat* $[p \cap g,\ R,\ G,\ q,a]]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Guard f g c) sat* $[p \cap g,\ R,\ G,\ q,a]$

| *Guarantee*: $[\![$ *Sta p R*; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$; $f{\in}F$;
$\Gamma,\Theta \vdash_{/F}$ *c sat* $[p \cap g,\ R,\ G,\ q,a]\ ]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Guard f g c) sat* $[p,\ R,\ G,\ q,a]$

| *CallRec*: $[\![(c,p,R,G,q,a) \in Specs$;
$\forall (c,p,R,G,q,a){\in}\ Specs.\ c \in dom\ \Gamma\ \wedge$
Sta p R $\wedge\ (\forall s.\ (Normal\ s, Normal\ s) \in G) \wedge$
$\Gamma,\Theta \cup Specs \vdash_{/F}$ *(the* $(\Gamma\ c)$*) sat* $[p,\ R,\ G,\ q,a]$;
Sta p R; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ *(Call c) sat* $[p,\ R,\ G,\ q,a]$

| *Asm*: $[\![(c,p,R,G,q,a) \in \Theta]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Call c) sat* $[p,\ R,\ G,\ q,a]$

| *Call*: $[\![$
Sta p R; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$;$c \in dom\ \Gamma$;
$\Gamma,\Theta \vdash_{/F}$ *(the* $(\Gamma\ c)$*) sat* $[p,\ R,\ G,\ q,a]]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Call c) sat* $[p,\ R,\ G,\ q,a]$

| *DynCom*: $[\![$(*Sta p R*) $\wedge$ (*Sta q R*) $\wedge$ (*Sta a R*) $\wedge$
$(\forall s.\ (Normal\ s, Normal\ s) \in G)$;
$(\forall s \in p.\ (\Gamma,\Theta\vdash_{/F}$ $(c\ s)$ *sat* $[p,\ R,\ G,\ q,a]))]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ *(DynCom c) sat* $[p,\ R,\ G,\ q,a]$

| *Throw*: $[\![$*Sta a R*; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)\ ]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *Throw sat* $[a,\ R,\ G,\ q,a]$

| *Catch*: $[\![$*Sta q R*; $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$;
$\Gamma,\Theta \vdash_{/F}$ *c1 sat* $[p,\ R,\ G,\ q,r]$;
$\Gamma,\Theta \vdash_{/F}$ *c2 sat* $[r,\ R,\ G,\ q,a]]\!] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *(Catch c1 c2) sat* $[p,\ R,\ G,\ q,a]$

| *Conseq*: $\forall\, s \in p.$
$(\exists\, p'\ R'\ G'\ q'\ a'\ \Theta'.$
$(s\in p') \wedge$
$R \subseteq R' \wedge$
$G' \subseteq G \wedge$
$q' \subseteq q \wedge$
$a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$
$(\Gamma,\Theta' \vdash_{/F} P\ sat\ [p',\ R',\ G',\ q',a'])\ )$
$\Longrightarrow \Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q,a]$

| *Conj-post*: $\Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q,a] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q',a']$
$\Longrightarrow \Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q \cap q',a \cap a']$

| *Conj-Inter*: $sa \neq (\{\}::nat\ set) \Longrightarrow$
$\forall\, i{\in}sa.\ \Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q\ i,a] \Longrightarrow$
$\Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\bigcap\, i{\in}sa.\ q\ i,a]$

inductive-cases *hoare-elim-cases* [*cases set*]:
$\Gamma,\Theta \vdash_{/F} Skip\ sat\ [p,\ R,\ G,\ q,a]$

thm *hoare-elim-cases*

definition *Pre* :: $('s,'p,'f,'e)rgformula \Rightarrow ('s\ set)$ **where**
$Pre\ x \equiv fst(snd\ x)$

definition *Post* :: $('s,'p,'f,'e)\ rgformula \Rightarrow ('s\ set)$ **where**
$Post\ x \equiv\ fst(snd(snd(snd(snd\ x))))$

definition *Abr* :: $('s,'p,'f,'e)\ rgformula \Rightarrow ('s\ set)$ **where**
$Abr\ x \equiv snd(snd(snd(snd(snd\ x))))$

definition *Rely* :: $('s,'p,'f,'e)\ rgformula \Rightarrow (('s,'f)\ tran)\ set$ **where**
$Rely\ x \equiv fst(snd(snd\ x))$

definition *Guar* :: $('s,'p,'f,'e)\ rgformula \Rightarrow (('s,'f)\ tran)\ set$ **where**
$Guar\ x \equiv fst(snd(snd(snd\ x)))$

definition *Com* :: $('s,'p,'f,'e)\ rgformula \Rightarrow ('s\ ,'p,'f,'e)\ com$ **where**
$Com\ x \equiv fst\ \ x$

inductive

par-rghoare :: $[('s,'p,'f,'e)$ *body*,
$('s,'p,'f,'e)$ *sextuple set*,
$'f$ *set*,
$(\ ('s,'p,'f,'e)$ *rgformula*$)$ *list*,
$'s$ *set*,
$(('s,'f)$ *tran*$)$ *set*, $(('s,'f)$ *tran*$)$ *set*,
$'s$ *set*,
$'s$ *set*$] \Rightarrow$ *bool*
$(\text{-,-} \vdash_{'/\text{-}}$ - *SAT* [-, -, -, -,-] [61,60,60,0,0,0,0] 45)

where

Parallel:
$[\![\ \forall i{<}length\ xs.\ R \cup (\bigcup j{\in}\{j.\ j{<}length\ xs \wedge j{\neq}i\}.\ (Guar(xs!j))) \subseteq (Rely(xs!i));$
$(\bigcup j{<}length\ xs.\ (Guar(xs!j))) \subseteq G;$
$p \subseteq (\bigcap i{<}length\ xs.\ (Pre(xs!i)));$
$(\bigcap i{<}length\ xs.\ (Post(xs!i))) \subseteq q;$
$(\bigcup i{<}length\ xs.\ (Abr(xs!i))) \subseteq a;$
$\forall i{<}length\ xs.\ \Gamma,\Theta \vdash_{/F} Com(xs!i)\ sat\ [Pre(xs!i),Rely(xs!i),Guar(xs!i),Post(xs!i),Abr(xs!i)]$
$]\!]$
$\Longrightarrow\ \Gamma,\Theta \vdash_{/F} xs\ SAT\ [p,\ R,\ G,\ q,a]$

12 Soundness

lemma *skip-suc-i*:
assumes *a1*:$(\Gamma,\ l) \in cptn \wedge fst\ (l!i) = Skip$
assumes *a2*:$i{+}1 < length\ l$
shows $fst\ (l!(i{+}1)) = Skip$
proof −
from *a2 a1* **obtain** *l1 ls* **where** $l{=}l1\#ls$
by (*metis list.exhaust list.size*(*3*) *not-less0*)
then have $\Gamma{\vdash_c}\ (l!i) \rightarrow_{ce} (l!(Suc\ i))$ **using** *cptn-stepc-rtran a1 a2*
by *fastforce*
thus *?thesis* **using** *a1 a2 step-ce-elim-cases*
by (*metis* (*no-types*) *Suc-eq-plus1 not-eq-not-env prod.collapse stepc-elim-cases*(*1*))
qed

lemma *throw-suc-i*:
assumes *a1*:$(\Gamma,\ l) \in cptn \wedge (fst(l!i) = Throw \wedge snd(l!i) = Normal\ s1)$
assumes *a2*:$Suc\ i < length\ l$
assumes *a3*:$env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ q\ rely \wedge s1 \in q$
shows $fst\ (l!(Suc\ i)) = Throw \wedge (\exists s2.\ snd(l!(Suc\ i)) = Normal\ s2 \wedge s2 \in q)$
proof −
have *fin*:*final* $(l!i)$ **using** *a1* **unfolding** *final-def* **by** *auto*
from *a2 a1* **obtain** *l1 ls* **where** $l{=}l1\#ls$
by (*metis list.exhaust list.size*(*3*) *not-less0*)
then have $\Gamma{\vdash_c}\ (l!i) \rightarrow_{ce} (l!(Suc\ i))$ **using** *cptn-stepc-rtran a1 a2*
by *fastforce* **then have** $\Gamma{\vdash_c}\ (l!i) \rightarrow (l!(Suc\ i)) \vee \Gamma{\vdash_c}\ (l!i) \rightarrow_e (l!(Suc\ i))$
using *step-ce-elim-cases* **by** *blast*
thus *?thesis* **proof**

assume $\Gamma\vdash_c (l!i) \rightarrow (l!(Suc\ i))$ **thus** *?thesis* **using** *fin no-step-final'* **by** *blast*
next
assume $\Gamma\vdash_c (l!i) \rightarrow_e (l!(Suc\ i))$ **thus** *?thesis*
using *a1 a3 a2 env-tran-normal* **by** (*metis* (*no-types*, *lifting*) *env-c-c' prod.collapse*)
qed
qed

lemma *i-skip-all-skip*:**assumes** *a1*:$(\Gamma, l) \in cptn \wedge fst\ (l!i) = Skip$
assumes *a2*: $i\leq j \wedge j < (length\ l)$
assumes *a3*:$n=j-i$

shows $fst\ (l!j) = Skip$
using *a1 a2 a3*
proof (*induct n arbitrary*: *i j*)
case *0*
then have $Suc\ i = Suc\ j$ **by** *simp*
thus *?case* **using** *0.prems skip-suc-i* **by** *fastforce*
next
case (*Suc n*)
then have $length\ l > Suc\ i$ **by** *auto*
then have $i<j$ **using** *Suc* **by** *fastforce*
moreover then have $j-1< length\ l$ **using** *Suc* **by** *fastforce*
moreover then have $j - i = Suc\ n$ **using** *Suc* **by** *fastforce*
ultimately have $fst\ (l\ !\ (j)) = LanguageCon.com.Skip$ **using** *Suc skip-suc-i*
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc Suc-eq-plus1 Suc-leI* ‹$Suc\ i < length\ l$› *diff-Suc-1*)
also have $j=j$ **using** *Cons* **using** *Suc.prems*(*2*) **by** *linarith*
ultimately show *?case* **using** *Suc* **by** (*metis* (*no-types*))
qed

lemma *i-throw-all-throw*:**assumes** *a1*:$(\Gamma, l) \in cptn \wedge (fst\ (l!i) = Throw \wedge snd\ (l!i) = Normal\ s1)$
assumes *a2*: $i\leq j \wedge j < (length\ l)$
assumes *a3*:$n=j-i$
assumes *a4*:$env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ q\ rely \wedge s1\in q$
shows $fst\ (l!j) = Throw \wedge (\exists s2.\ snd(l!j) = Normal\ s2 \wedge s2\in q)$
using *a1 a2 a3 a4*
proof (*induct n arbitrary*: *i j s1*)
case *0*
then have $Suc\ i = Suc\ j$ **by** *simp*
thus *?case* **using** *0.prems skip-suc-i* **by** *fastforce*
next
case (*Suc n*)
then have *l-suc*:$length\ l > Suc\ i$ **by** *linarith*
then have $i<j$ **using** *Suc.prems*(*3*) **by** *linarith*
moreover then have $j-1< length\ l$ **by** (*simp add*: *Suc.prems*(*2*) *less-imp-diff-less*)

moreover then have $j - Suc\ i = n$ **by** (*metis Suc-diff-Suc Suc-inject* ‹$i < j$›

Suc(*4*))
ultimately obtain *s2* **where** *fst* (*l* ! (*j*−*1*)) = *LanguageCon.com.Throw* ∧ *snd* (*l* ! (*j*−*1*)) = *Normal s2* ∧ *s2*∈*q*
using *Suc*(*1*)[*of i s1 j*−*1*] *Suc*(*2*) *Suc*(*5*)
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc diff-Suc-eq-diff-pred diff-zero less-imp-Suc-add not-le not-less-eq-eq zero-less-Suc*)
also have *Suc* (*j* − *1*) < *length l* **using** *Suc* **by** *arith*
ultimately have *fst* (*l* ! (*j*)) = *LanguageCon.com.Throw* ∧ (∃ *s2*. *snd*(*l*!*j*) = *Normal s2* ∧ *s2*∈*q*)
using *Suc*(*2*−*5*) *throw-suc-i*[*of* Γ *l j*−*1 s2 rely q*] *a4*
by *fastforce*
also have *j*=*j* **using** *Cons* **using** *Suc.prems*(*2*) **by** *linarith*
ultimately show *?case* **using** *Suc* **by** (*metis* (*no-types*))
qed

lemma *only-one-component-tran-j*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*i*) = *Skip* ∨ *fst* (*l*!*i*) = *Throw* **and**
a1′: *snd* (*l*!*i*) = *Normal x* ∧ *x* ∈ *q* **and**
a2: *i*≤*j* ∧ *Suc j* < *length l* **and**
a3: ($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*))) **and**
a4: *env-tran-right* Γ *l rely* ∧ *Sta q rely*
shows *P*
proof −
have *fst* (*l*!*j*) = *Skip* ∨ (*fst* (*l*!*i*) = *Throw* ∧ *snd*(*l*!*i*) = *Normal x*)
using *a0 a1 a1′ a2 a3 a4 i-skip-all-skip* **by** *fastforce*
also have ($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*))) **using** *a3* **by** *fastforce*
ultimately show *?thesis*
by (*meson SmallStepCon.final-def SmallStepCon.no-step-final′ Suc-lessD a0 a2 a4 i-throw-all-throw a1′*)
qed

lemma *only-one-component-tran-all-j*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*i*) = *Skip* ∨ (*fst* (*l*!*i*) = *Throw* ∧ *snd*(*l*!*i*) = *Normal s1*) **and**
a1′: *snd* (*l*!*i*) = *Normal x* ∧ *x* ∈ *q* **and**
a2: *Suc i*<*length l* **and**
a3: ∀ *j*. *i*≤*j* ∧ *Suc j* < *length l* ⟶ ($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*))) **and**
a4: *env-tran-right* Γ *l rely* ∧ *Sta q rely*
shows *P*
using *a0 a1 a2 a3 a4 a1′ only-one-component-tran-j*
by (*metis lessI less-Suc-eq-le*)

lemma *zero-skip-all-skip*:
assumes *a1*:(Γ, *l*) ∈ *cptn* ∧ *fst* (*l*!*0*) = *Skip* ∧ *i* < *length l*
shows *fst* (*l*!*i*) = *Skip*
using *a1 i-skip-all-skip* **by** *blast*

lemma *all-skip*:
assumes
a0:$(\Gamma,x)\in cptn$ **and**
a1:$x!0 = (Skip,s)$
shows $(\forall i<length\ x.\ fst(x!i) = Skip)$
using *a0 a1 zero-skip-all-skip* **by** *fastforce*

lemma *zero-throw-all-throw*:
assumes *a1*:$(\Gamma,\ l) \in cptn \wedge fst\ (l!0) = Throw \wedge$
$snd(l!0) = Normal\ s1 \wedge\ \ i < length\ l \wedge s1\in q$
assumes *a2*: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ q\ rely$
shows $fst\ (l!i) = Throw \wedge (\exists s2.\ snd\ (l!i) = Normal\ s2)$
using *a1 a2 i-throw-all-throw* **by** (*metis le0*)

lemma *only-one-component-tran-0*:
assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
a1: $(fst\ (l!0) = Skip) \vee (fst\ (l!0) = Throw)$ **and**
a1': $snd\ (l!0) = Normal\ x \wedge x \in q$ **and**
a2: $Suc\ j < length\ l$ **and**
a3: $(\Gamma\vdash_c(l!j)\ \rightarrow (l!(Suc\ j)))$ **and**
a4: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ q\ rely$
shows P
proof−
have *a2'*:$0\le j \wedge Suc\ j<length\ l$ **using** *a2* **by** *arith*
show *?thesis*
using *only-one-component-tran-j*[*OF a0 a1 a1' a2' a3 a4*] **by** *auto*
qed

lemma *not-step-comp-step-env*:
assumes *a0*: $(\Gamma,\ l) \in cptn$ **and**
a1: $(Suc\ j<length\ l)$ **and**
a2: $(\forall k < j.\ \neg((\Gamma\vdash_c(l!k)\ \rightarrow (l!(Suc\ k)))))$
shows $(\forall k < j.\ ((\Gamma\vdash_c(l!k)\ \rightarrow_e (l!(Suc\ k)))))$
proof −
{**fix** k
assume *asm*: $k<j$
also then have $Suc\ k<length\ l$ **using** *a1 a2* **by** *auto*
ultimately have $(\Gamma\vdash_c(l!k)\ \rightarrow_{ce} (l!(Suc\ k)))$ **using** *a0 cptn-stepc-rtran*
proof −
obtain $nn :: nat \Rightarrow nat \Rightarrow nat$ **where**
f1: $\forall x0\ x1.\ (\exists v2>x1.\ x0 = Suc\ v2) = (x1 < nn\ x0\ x1 \wedge x0 = Suc\ (nn\ x0\ x1))$
by *moura*
obtain $pp :: nat \Rightarrow (('b,\ 'a,\ 'c,'d)\ LanguageCon.com \times ('b,\ 'c)\ xstate)\ list \Rightarrow ('b,\ 'a,\ 'c,'d)\ LanguageCon.com \times ('b,\ 'c)\ xstate$ **and**
$pps :: nat \Rightarrow (('b,\ 'a,\ 'c,'d)\ LanguageCon.com \times ('b,\ 'c)\ xstate)\ list \Rightarrow (('b,\ 'a,\ 'c,'d)\ LanguageCon.com \times ('b,\ 'c)\ xstate)\ list$ **where**
$\forall x0\ x1.\ (\exists v2\ v3.\ x1 = v2\ \#\ v3 \wedge length\ v3 = x0) = (x1 = pp\ x0\ x1\ \#\ pps\ x0\ x1 \wedge length\ (pps\ x0\ x1) = x0)$

by *moura*
then have *f2*: $l = pp\ (nn\ (length\ l)\ k)\ l\ \#\ pps\ (nn\ (length\ l)\ k)\ l \wedge length\ (pps\ (nn\ (length\ l)\ k)\ l) = nn\ (length\ l)\ k$
using *f1* **by** (*meson Suc-lessE* ‹*Suc k* < *length l*› *length-Suc-conv*)
then have *f3*: $Suc\ k < length\ (pp\ (nn\ (length\ l)\ k)\ l\ \#\ pps\ (nn\ (length\ l)\ k)\ l)$
by (*metis* ‹*Suc k* < *length l*›)
have $(\Gamma,\ pp\ (nn\ (length\ l)\ k)\ l\ \#\ pps\ (nn\ (length\ l)\ k)\ l) \in cptn$
using *f2 a0* **by** *presburger*
then have $\Gamma\vdash_c (pp\ (nn\ (length\ l)\ k)\ l\ \#\ pps\ (nn\ (length\ l)\ k)\ l)\ !\ k \rightarrow_{ce} (pp\ (nn\ (length\ l)\ k)\ l\ \#\ pps\ (nn\ (length\ l)\ k)\ l)\ !\ Suc\ k$
using *f3* **by** (*meson cptn-stepc-rtran*)
then show *?thesis*
using *f2* **by** *auto*
qed
also have $\neg((\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k))))$ **using** *a2 asm* **by** *auto*
ultimately have $((\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k))))$ **using** *step-ce-elim-cases* **by** *blast*
} **thus** *?thesis* **by** *auto*
qed

lemma *cptn-i-env-same-prog*:
assumes *a0*: $(\Gamma,\ l) \in cptn$ **and**
a1: $\forall k < j.\ k\geq i \longrightarrow (\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k)))$ **and**
a2: $i\leq j \wedge j < length\ l$
shows $fst\ (l!j) = fst\ (l!i)$
using *a0 a1 a2*
proof (*induct j−i arbitrary*: *l j i*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc n*)
then have *lenl*:$length\ l>Suc\ 0$ **by** *fastforce*
have $j>0$ **using** *Suc* **by** *linarith*
then obtain *j1* **where** *prev*:$j=Suc\ j1$
using *not0-implies-Suc* **by** *blast*
then obtain *a0 a1 l1* **where** *l*:$l=a0\#l1@[a1]$
using *Suc lenl* **by** (*metis add.commute add.left-neutral length-Cons list.exhaust list.size(3) not-add-less1 rev-exhaust*)
then have *al1-cptn*:$(\Gamma,a0\#l1)\in cptn$
using *Suc.prems(1) Suc.prems(3) tl-in-cptn cptn-dest-2*
by *blast*
have *i-j*:$i\leq j1$ **using** *Suc prev* **by** *auto*
have $\forall k < j1.\ k\geq i \longrightarrow (\Gamma\vdash_c((a0\#l1)!k) \rightarrow_e ((a0\#l1)!(Suc\ k)))$
proof −
{**fix** *k*
assume *a0*:$k<j1 \wedge k\geq i$
then have $(\Gamma\vdash_c((a0\#l1)!k) \rightarrow_e ((a0\#l1)!(Suc\ k)))$
using *l Suc(4) prev lenl Suc(5)*
proof −
have *suc-k-j*:$Suc\ k < j$ **using** *a0 prev* **by** *blast*

```
        have j1-l-l1:j1 < Suc (length l1)
          using Suc.prems(3) l prev by auto
        have k < Suc j1
          using ‹k < j1 ∧ i ≤ k› less-Suc-eq by blast
        hence f3: k < j
          using prev by blast
        hence ksuc:k < Suc (Suc j1)
          using less-Suc-eq prev by blast
        hence f4: k < Suc (length l1)
          using  prev Suc.prems(3) l a0 j1-l-l1 less-trans
          by blast
        have f6: Γ⊢c l ! k →e (l ! Suc k)
          using f3 Suc(4) a0 by blast
        have k-l1:k < length l1
          using f3 Suc.prems(3) i-j l suc-k-j  by auto
        thus ?thesis
        proof (cases k)
          case 0 thus ?thesis using f6 l  k-l1
              by (simp add: nth-append)
        next
          case (Suc k1) thus ?thesis
            using f6  f4 l k-l1
            by (simp add: nth-append)
        qed
      qed
     }thus ?thesis by auto
   qed
   then have fst:fst ((a0#l1)!i)=fst ((a0#l1)!j1)
     using Suc(1)[of j1 i a0#l1]
           Suc(2) Suc(3) Suc(4) Suc(5) prev al1-cptn i-j
     by (metis (mono-tags, lifting) Suc-diff-le Suc-less-eq diff-Suc-1 l length-Cons
length-append-singleton)
   have len-l:length l = Suc (length (a0#l1)) using l by auto
   then have f1:i<length (a0#l1) using Suc.prems(3) i-j prev by linarith
   then have f2:j1<length (a0#l1) using Suc.prems(3) len-l prev by auto
   have i-l:fst (l!i) = fst ((a0#l1)!i)
     using l prev f1 f2 fst
     by (metis (no-types) append-Cons nth-append)
   also have j1-l:fst (l!j1) = fst ((a0#l1)!j1)
   using l prev f1 f2 fst
     by (metis (no-types) append-Cons nth-append)
   then have fst (l!i) = fst (l!j1) using
     i-l j1-l fst by auto
   thus ?case using Suc prev by (metis env-c-c' i-j lessI prod.collapse)
qed


lemma cptn-tran-ce-i:
  assumes a1:(Γ, l) ∈ cptn  ∧  i + 1 < length l
```

shows $\Gamma \vdash_c (l!i) \ \rightarrow_{ce} \ (l!(Suc\ i))$
proof −
from *a1*
obtain *a1 l1* **where** *l=a1#l1* **using** *cptn.simps* **by** *blast*
thus *?thesis* **using** *a1 cptn-stepc-rtran* **by** *fastforce*
qed

lemma *zero-final-always-env-0*:
assumes *a1*:$(\Gamma,\ l) \in cptn$ **and**
a2: $fst\ (l!0) = Skip \lor fst\ (l!0) = Throw$ **and**
a2': $snd(l!0) = Normal\ s1 \ \land\ s1 \in q$ **and**
a3: $Suc\ i < length\ l$ **and**
a4: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \land Sta\ q\ rely$
shows $\Gamma \vdash_c (l!i) \ \rightarrow_e \ (l!(Suc\ i))$
proof −
have $\Gamma \vdash_c (l!i) \ \rightarrow_{ce} \ (l!(Suc\ i))$ **using** *a1 a2 a3 cptn-tran-ce-i* **by** *auto*
also have $\neg\ (\Gamma \vdash_c (l!i) \ \rightarrow \ (l!(Suc\ i)))$ **using** *a1 a2 a3 a4 a2'*
using *only-one-component-tran-0* **by** *metis*
ultimately show *?thesis* **by** (*simp add*: *step-ce.simps*)
qed

lemma *final-always-env-i*:
assumes *a1*:$(\Gamma,\ l) \in cptn$ **and**
a2: $fst\ (l!0) = Skip \lor fst\ (l!0) = Throw$ **and**
a2': $snd(l!0) = Normal\ s1 \ \land\ s1 \in q$ **and**
a3: $j \geq i \land Suc\ j < length\ l$ **and**
a4: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \land Sta\ q\ rely$
shows $\Gamma \vdash_c (l!j) \ \rightarrow_e \ (l!(Suc\ j))$
proof −
have *ce-tran*:$\Gamma \vdash_c (l!j) \ \rightarrow_{ce} \ (l!(Suc\ j))$ **using** *a1 a2 a3 a4 cptn-tran-ce-i* **by** *auto*

then have $\Gamma \vdash_c (l!j) \ \rightarrow_e \ (l!(Suc\ j)) \lor \Gamma \vdash_c (l!j) \ \rightarrow \ (l!(Suc\ j))$
using *step-ce-elim-cases* **by** *blast*
thus *?thesis*
proof
assume $\Gamma \vdash_c (l!j) \ \rightarrow_e \ (l!(Suc\ j))$ **then show** *?thesis* **by** *auto*
next
assume *a01*:$\Gamma \vdash_c (l!j) \ \rightarrow \ (l!(Suc\ j))$
then have $\neg\ (\Gamma \vdash_c (l!j) \ \rightarrow \ (l!(Suc\ j)))$
using *a1 a2 a3 a4 a2' only-one-component-tran-j* [*OF a1*]
by *blast*
then show *?thesis* **using** *a01 ce-tran* **by** (*simp add*: *step-ce.simps*)
qed
qed

12.1 Skip Sound

lemma *stable-q-r-q*:
assumes *a0*:*Sta q R* **and**

$a1$: $snd(l!i) \in Normal$ ‘ q **and**
$a2$:$(snd(l!i), snd(l!(Suc\ i))) \in R$
shows $snd(l!(Suc\ i)) \in Normal$ ‘ q
using *a0 a1 a2*
unfolding *Sta-def* **by** *fastforce*

lemma *stability*:
assumes $a0$:*Sta q R* **and**
$a1$: $snd(l!j) \in Normal$ ‘ q **and**
$a2$: $j \leq k \wedge k < (length\ l)$ **and**
$a3$: $n = k - j$ **and**
$a4$: $\forall i.\ j \leq i \wedge i < k \longrightarrow \Gamma \vdash_c (l!i) \ \rightarrow_e (l!(Suc\ i))$ **and**
$a5$:*env-tran-right* Γ *l R*
shows $snd\ (l!k) \in Normal$ ‘ $q \wedge fst\ (l!j) = fst\ (l!k)$
using *a0 a1 a2 a3 a4 a5*
proof (*induct n arbitrary*: *j k*)
case *0*
thus *?case* **by** *auto*
next
case (*Suc n*)
then have $length\ l > j + 1$ **by** *arith*
moreover then have $k-1 < length\ l$ **using** *Suc* **by** *fastforce*
moreover then have $(k - 1) - j = n$ **using** *Suc* **by** *fastforce*
moreover then have $j \leq k-1$ **using** *Suc* **by** *arith*
moreover have $\forall i.\ j \leq i \wedge i < k-1 \longrightarrow \Gamma \vdash_c (l\ !\ i) \rightarrow_e (l\ !\ Suc\ i)$
using *Suc* **by** *fastforce*
ultimately have *induct*:$snd\ (l!\ (k-1)) \in Normal$ ‘ $q \wedge fst\ (l!j) = fst\ (l!(k-1))$ **using** *Suc*
by *blast*
also have *j-1*:$k-1+1=k$ **using** *Cons Suc.prems*(*4*) **by** *auto*
have *f1*:$\forall i.\ j \leq i \wedge i < k \longrightarrow (snd((snd\ (\Gamma,l))!i), snd((snd\ (\Gamma,l))!(Suc\ i))) \in R$
using *Suc* **unfolding** *env-tran-right-def* **by** *fastforce*
have *k1*:$k - 1 < k$
by (*metis* (*no-types*) *Suc-eq-plus1 j-1 lessI*)
then have $(snd((snd\ (\Gamma,l))!(k-1)), snd((snd\ (\Gamma,l))!(Suc\ (k-1)))) \in R$
using ‹$j \leq k - 1$› *f1* **by** *blast*
ultimately have $snd\ (l!k) \in Normal$ ‘ q **using** *stable-q-r-q Suc*(*2*) *Suc*(*5*) **by** *fastforce*
also have $fst\ (l!j) = fst\ (l!k)$
proof −
have $\Gamma \vdash_c (l\ !\ (k-1)) \rightarrow_e (l\ !\ k)$ **using** *Suc*(*6*) *k1* ‹$j \leq k-1$› **by** *fastforce*
thus *?thesis* **using** *k1 prod.collapse env-c-c'* *induct* **by** *metis*
qed
ultimately show *?case* **by** *meson*
qed

lemma *stable-only-env-i-j*:
assumes $a0$:*Sta q R* **and**

$a1$: $snd(l!i) \in Normal$ ‘ q **and**
$a2$: $i<j \wedge j < (length\ l)$ **and**
$a3$: $n=j-i-1$ **and**
$a4$: $\forall k \geq i.\ k < j \longrightarrow \Gamma \vdash_c (l!k) \rightarrow_e (l!(Suc\ k))$ **and**
$a5$: *env-tran-right* $\Gamma\ l\ R$
shows $snd\ (l!j) \in Normal$ ‘ q
using *a0 a1 a2 a3 a4 a5* **by** (*meson less-imp-le-nat stability*)

lemma *stable-only-env-1*:
assumes *a0*:*Sta q R* **and**
$a1$: $snd(l!i) \in Normal$ ‘ q **and**
$a2$: $i<j \wedge j < (length\ l)$ **and**
$a3$: $n=j-i-1$ **and**
$a4$: $\forall i.\ Suc\ i < length\ l \longrightarrow \Gamma \vdash_c (l!i) \rightarrow_e (l!(Suc\ i))$ **and**
$a5$: *env-tran-right* $\Gamma\ l\ R$
shows $snd\ (l!j) \in Normal$ ‘ q
using *a0 a1 a2 a3 a4 a5*
by (*meson stable-only-env-i-j less-trans-Suc*)

lemma *stable-only-env-q*:
assumes *a0*:*Sta q R* **and**
$a1$: $\forall i.\ Suc\ i < length\ l \longrightarrow \Gamma \vdash_c (l!i) \rightarrow_e (l!(Suc\ i))$ **and**
$a2$: *env-tran* $\Gamma\ q\ l\ R$
shows $\forall i.\ i < length\ l \longrightarrow snd\ (l!i) \in Normal$ ‘ q
proof (*cases* $0 < length\ l$)
case *False* **thus** *?thesis* **using** *a2* **unfolding** *env-tran-def* **by** *fastforce*
next
case *True*
thus *?thesis*
proof − {
fix *i*
assume *aa1*:$i < length\ l$
have *post-0*:$snd\ (l\ !\ 0) \in Normal$ ‘ q
using *a2* **unfolding** *env-tran-def* **by** *auto*
then have $snd\ (l\ !\ i) \in Normal$ ‘ q
proof (*cases i*)
case *0* **thus** *?thesis* **using** *post-0* **by** *auto*
next
case (*Suc n*)

have *env-tran-right* $\Gamma\ l\ R$
using *a2 env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
also have $0<i$ **using** *Suc* **by** *auto*
ultimately show *?thesis*
using *post-0 stable-only-env-1 a0 a1 a2 aa1* **by** *blast*
qed
} **then show** *?thesis* **by** *auto* **qed**

qed

lemma *Skip-sound1*:
assumes *a0*:*Sta q R* **and**
a1:(∀ *s*. (*Normal s*, *Normal s*) ∈ *G*) **and**
a10:*c* ∈ *cp* Γ *Skip s* **and**
a11:*c* ∈ *assum*(*q*, *R*)
shows *c* ∈ *comm* (*G*, (*q*,*a*)) *F*
proof −
obtain Γ*1 l* **where** *c-prod*:*c*=(Γ*1*,*l*) **by** *fastforce*
{
assume *snd* (*last l*) ∉ *Fault* ‘ *F*
have *cp*:*l*!*0*=(*Skip*,*s*) ∧ (Γ,*l*) ∈ *cptn* ∧ Γ=Γ*1* **using** *a10 cp-def c-prod* **by** *fastforce*
have *assum*:*snd*(*l*!*0*) ∈ *Normal* ‘ *q* ∧ (∀ *i*. *Suc i*<*length l* ⟶
(Γ*1*)⊢$_c$(*l*!*i*) →$_e$ (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *R*)
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
have *concl*:(∀ *i*. *Suc i*<*length l* ⟶
Γ*1*⊢$_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)
proof −
{ **fix** *i*
assume *asuc*:*Suc i*<*length l*
then have ¬ (Γ*1*⊢$_c$(*l*!*i*) → (*l*!(*Suc i*)))
by (*metis Suc-lessD cp prod.collapse prod.sel*(*1*) *stepc-elim-cases*(*1*) *zero-skip-all-skip*)
} **thus** *?thesis* **by** *auto* **qed**
have *concr*:(*final* (*last l*) ⟶
((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ‘ (*a*)))
proof−
{
assume *valid*:*final* (*last l*)
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a l1* **where** *l*:*l*=*a*#*l1* **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a l1*] *l* **by** *fastforce*
then have *fst-last-skip*:*fst* (*last l*) = *Skip*
by (*metis* ‹*0* < *length l*› *cp diff-less fst-conv zero-less-one zero-skip-all-skip*)

have *last-q*: *snd* (*last l*) ∈ *Normal* ‘ *q*
proof −
have *env*: *env-tran* Γ *q l R* **using** *env-tran-def assum cp* **by** *blast*
have *env-right*:*env-tran-right* Γ *l R* **using** *a0 env-tran-right-def assum cp* **by** *metis*

also obtain *s1* **where** $snd(l!0) = Normal\ s1 \wedge s1{\in}q$
using *assum* **by** *auto*
ultimately have *all-tran-env*: $\forall i.\ Suc\ i < length\ l \longrightarrow \Gamma\vdash_c(l!i)\ \rightarrow_e\ (l!(Suc\ i))$
using *final-always-env-i cp zero-final-always-env-0 a0*
by *fastforce*
then have $\forall i.\ i < length\ l \longrightarrow snd\ (l!i) \in Normal\ `\ q$
using *stable-only-env-q a0 env* **by** *fastforce*
thus *?thesis* **using** *last-l* **using** *len-l* **by** *fastforce*
qed
note *res* = *conjI* [*OF fst-last-skip last-q*]
} **thus** *?thesis* **by** *auto*
qed
note *res* = *conjI* [*OF concl concr*]
} **thus** *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *auto*
qed

lemma *Skip-sound*:
Sta q R $\Longrightarrow$
$(\forall s.\ (Normal\ s,\ Normal\ s) \in G) \Longrightarrow$
$\Gamma,\Theta \models n_{/F}$ *Skip sat* [*q*,*R*, *G*, *q*,*a*]
proof−
assume
a0:*Sta q R* **and**
a1:$(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$
{
fix *s*
have *ass*:*cpn n* Γ *Skip s* $\cap$ *assum*(*q*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
proof−
{ **fix** *c*
assume *a10*:*c* $\in$ *cpn n* Γ *Skip s* **and** *a11*:*c* $\in$ *assum*(*q*, *R*)
then have *a10*:*c*$\in$*cp* Γ *Skip s*
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
have *c*$\in$*comm*(*G*, (*q*,*a*)) *F* **using** *Skip-sound1*[*OF a0 a1 a10 a11*] **by** *auto*

} **thus** *?thesis* **by** *auto*
qed
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Throw-sound1*:
assumes *a1*:*Sta a R* **and**
a2:$(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$ **and**
a10:*c* $\in$ *cp* Γ *Throw s* **and**
a11:*c* $\in$ *assum*(*a*, *R*)
shows *c* $\in$ *comm* (*G*, (*q*,*a*)) *F*
proof −

obtain $\Gamma 1$ *l* **where** *c-prod*:$c=(\Gamma 1,l)$ **by** *fastforce*
{
assume *snd* (*last l*) $\notin$ *Fault* ' *F*
have *cp*:$l!0=(Throw,s) \wedge (\Gamma,l) \in cptn \wedge \Gamma=\Gamma 1$ **using** *a10 cp-def c-prod* **by** *fastforce*
have *assum*:$snd(l!0) \in Normal$ ' $(a) \wedge (\forall i.\ Suc\ i<length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in (R))$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:*env-tran-right* Γ *l R* **using** *cp env-tran-right-def* **by** *auto*
obtain *a1* **where** *a-normal*:$snd(l!0) = Normal\ a1 \wedge a1 \in a$
using *assum* **by** *auto*
have *concl*:$(\forall i\ ns\ ns'.\ Suc\ i<length\ l \longrightarrow$
$\Gamma 1\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in\ (G))$
proof –
{ fix *i*
assume *asuc*:$Suc\ i<length\ l$
then have *asuci*:$i<length\ l$ **by** *fastforce*
then have *fst* $(l\ !\ 0) = LanguageCon.com.Throw$ **using** *cp* **by** *auto*
moreover obtain *s1* **where** *snd* $(l\ !\ 0) = Normal\ s1$ **using** *assum* **by** *auto*
ultimately have *fst* $(l\ !\ i) = Throw \wedge (\exists s2.\ snd\ (l\ !\ i) = Normal\ s2)$
using *cp a1 assum a-normal env-tran asuci zero-throw-all-throw*
by *fastforce*
then have $\neg\ (\Gamma 1\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)))$
by (*meson SmallStepCon.final-def SmallStepCon.no-step-final'*)
} thus *?thesis* **by** *auto* **qed**
have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ' *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ' (*a*)))
proof–
{
assume *valid*:*final* (*last l*)
have *len-l*:$length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:$l=a1\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:$last\ l = l!(length\ l-1)$
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
then have *fst-last-skip*:*fst* (*last l*) = *Throw*
by (*metis a1 a-normal cp diff-less env-tran fst-conv len-l zero-less-one zero-throw-all-throw*)
have *last-q*: *snd* (*last l*) $\in$ *Normal* ' (*a*)
proof –
have *env*: *env-tran* Γ *a l R* **using** *env-tran-def assum cp* **by** *blast*
have *env-right*:*env-tran-right* Γ *l R* **using** *env-tran-right-def assum cp* **by** *metis*
then have *all-tran-env*: $\forall i.\ Suc\ i < length\ l \longrightarrow \Gamma\vdash_c(l!i)\ \rightarrow_e\ (l!(Suc\ i))$
using *final-always-env-i a1 assum cp zero-final-always-env-0* **by** *fastforce*

then have $\forall i.\ i < \mathit{length}\ l \longrightarrow \mathit{snd}\ (l!i) \in \mathit{Normal}\ `\ (a)$
using *stable-only-env-q a1 env* **by** *fastforce*
thus *?thesis* **using** *last-l* **using** *len-l* **by** *fastforce*
qed
note *res* = *conjI* [*OF fst-last-skip last-q*]
} **thus** *?thesis* **by** *auto* **qed**
note *res* = *conjI* [*OF concl concr*]
}
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *auto*
qed

lemma *Throw-sound*:
Sta a R $\Longrightarrow$
$(\forall s.\ (\mathit{Normal}\ s,\ \mathit{Normal}\ s) \in G) \Longrightarrow$
$\Gamma,\Theta \models n_{/F}$ *Throw sat* [*a*, *R*, *G*, *q*,*a*]
proof −
assume
a1:*Sta a R* **and**
a2: $(\forall s.\ (\mathit{Normal}\ s,\ \mathit{Normal}\ s) \in G)$ **{**
fix *s*
have *ass*:*cpn n* Γ *Throw s* $\cap$ *assum*(*a*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
proof−
{ **fix** *c*
assume *a10*:*c* $\in$ *cpn n* Γ *Throw s* **and** *a11*:*c* $\in$ *assum*(*a*, *R*)
then have *a10*:*c*$\in$*cp* Γ *Throw s*
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
have *c*$\in$*comm*(*G*, (*q*,*a*)) *F* **using** *Throw-sound1*[*OF a1 a2 a10 a11*] **by** *auto*

} **thus** *?thesis* **by** *auto*
qed
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *no-comp-tran-before-i-0-g*:
assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
a1: *fst* (*l*!*0*) = *c* **and**
a2: *Suc i*<*length l* $\wedge$ $(\Gamma\vdash_c(l!i) \rightarrow (l!(\mathit{Suc}\ i)))$ **and**
a3: $j < i \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(\mathit{Suc}\ j)))$ **and**
a4: $\forall k < j.\ (\Gamma\vdash_c(l!k) \rightarrow_e (l!(\mathit{Suc}\ k)))$ **and**
a5: $\forall s1\ s2\ c1.\ \Gamma\vdash_c(c,\ s1) \rightarrow ((c1,s2)) \longrightarrow$
$(c1{=}\mathit{Skip}) \vee (c1{=}\mathit{Throw} \wedge (\exists s21.\ s2 = \mathit{Normal}\ s21))$ **and**

a6: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*0*) $\in$ *Normal* ‘ *p* $\wedge$
Sta q rely $\wedge$ *snd* (*l*!*Suc j*) $\in$ *Normal* ‘ *q*
shows *P*
proof −
have *Suc j* < *length l* **using** *a0 a1 a2 a3 a4* **by** *fastforce*

then have *fst* $(l!j) = c$
using *a0 a1 a2 a3 a4 cptn-env-same-prog*[*of* Γ *l j*] **by** *fastforce*
then obtain *s s1 c1* **where** *l-0*: $l!j = (c, s) \wedge l!(Suc\ j) = (c1,s1)$
by (*metis* (*no-types*) *prod.collapse*)
moreover have *snd* $(l!j) \in$ *Normal* ' *p* **using** *a4 stability*[*of p rely l 0 j j*] *a6 a3 a2*
proof −
have $\forall B\ r\ ps\ n\ na\ nb\ f.\ \neg$ *Sta B r* $\vee$ *snd* $(ps\ !\ n) \notin$ *Normal* ' $B \vee \neg\ n \leq na \vee \neg\ na <$ *length ps* $\vee\ na - n \neq nb \vee (\exists nb \geq n.\ nb < na \wedge \neg\ f \vdash_c ps\ !\ nb \rightarrow_e ps\ !\ Suc\ nb) \vee \neg$ *env-tran-right f ps r* $\vee$ *snd* $(ps\ !\ na) \in$ *Normal* ' $B \wedge$ (*fst* $(ps\ !\ n)$::('b, 'a, 'c,'d) *LanguageCon.com*) = *fst* $(ps\ !\ na)$
using *stability* **by** *blast*
then show *?thesis*
using *Suc-lessD* ‹*Suc j* < *length l*› *a4 a6* **by** *blast*
qed
then have *suc-0-skip*: (*fst* $(l!Suc\ j)$ = *Skip* $\vee$ *fst* $(l!Suc\ j)$ = *Throw*) $\wedge$
$(\exists s2.\ snd(l!Suc\ j) = Normal\ s2\ )$
using *a5 a6 a3 SmallStepCon.step-Stuck-prop* **using** *fst-conv imageE l-0 snd-conv* **by** *auto*
thus *?thesis* **using** *only-one-component-tran-j*
proof −
have $\forall n\ na.\ \neg\ n < na \vee Suc\ n \leq na$
using *Suc-leI* **by** *satx*
thus *?thesis* **using** *only-one-component-tran-j*[*OF a0*] *suc-0-skip a6 a0 a2 a3*
using *imageE* **by** *blast*
qed
qed

lemma *no-comp-tran-before-i*:
assumes *a0*:$(\Gamma, l) \in$ *cptn* **and**
a1: *fst* $(l!k) = c$ **and**
a2: *Suc i*<*length l* $\wedge$ $k \leq i \wedge (\Gamma \vdash_c (l!i)\ \rightarrow (l!(Suc\ i)))$ **and**
a3: $k \leq j \wedge j < i \wedge (\Gamma \vdash_c (l!j)\ \rightarrow (l!(Suc\ j)))$ **and**
a4: $\forall k < j.\ (\Gamma \vdash_c (l!k)\ \rightarrow_e (l!(Suc\ k)))$ **and**
a5: $\forall s1\ s2\ c1.\ \Gamma \vdash_c (c,\ s1)\ \rightarrow ((c1,s2)) \longrightarrow$
$(c1{=}Skip) \vee (c1{=}Throw \wedge (\exists s21.\ s2 = Normal\ s21))$ **and**

a6: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!0) \in$ *Normal* ' *p* $\wedge$
Sta q rely $\wedge$ *snd* $(l!Suc\ j) \in$ *Normal* ' *q*
shows *P*
using *a0 a1 a2 a3 a4 a5 a6*
proof (*induct k arbitrary*: *l i j*)
case *0* **thus** *?thesis* **using** *no-comp-tran-before-i-0-g* **by** *blast*
next
case (*Suc n*)
then obtain *a1 l1* **where** *l*: *l*=*a1*#*l1*
by (*metis less-nat-zero-code list.exhaust list.size*(*3*))
then have *l1notempty*:*l1*≠[] **using** *Suc* **by** *force*
then obtain *i'* **where** *i'*: *i*=*Suc i'* **using** *Suc*

using *less-imp-Suc-add* **by** *blast*
then obtain j' **where** j': $j=Suc\ j'$ **using** *Suc*
using *Suc-le-D* **by** *blast*
have $(\Gamma,l1)\in cptn$ **using** *Suc l*
using *tl-in-cptn l1notempty* **by** *blast*
moreover have $fst\ (l1\ !\ n) = c$
using *Suc l l1notempty* **by** *force*
moreover have $Suc\ i' < length\ l1 \wedge n \leq i' \wedge \Gamma\vdash_c l1\ !\ i' \rightarrow (l1\ !\ Suc\ i')$
using *Suc l l1notempty i'* **by** *auto*
moreover have $n \leq j' \wedge j' < i' \wedge \Gamma\vdash_c l1\ !\ j' \rightarrow (l1\ !\ Suc\ j')$
using *Suc l l1notempty i' j'* **by** *auto*
moreover have $\forall k<j'.\ \Gamma\vdash_c l1\ !\ k \rightarrow_e (l1\ !\ Suc\ k)$
using *Suc l l1notempty j'* **by** *auto*
moreover have *env-tran-right* Γ *l1 rely* $\wedge$ *Sta q rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l1!0)$ $\in$ *Normal ‘ p* $\wedge$
Sta q rely $\wedge$ *snd* $(l1!Suc\ j') \in$ *Normal ‘ q*
proof −
have *suc0*:$Suc\ 0 < length\ l$ **using** *Suc* **by** *auto*
have $j>0$ **using** j' **by** *auto*
then have $\Gamma\vdash_c(l!0) \rightarrow_e (l!(Suc\ 0))$ **using** *Suc*(*6*) **by** *blast*
then have $(snd(l!Suc\ 0) \in$ *Normal ‘ p*)
using *Suc*(*8*) *suc0* **unfolding** *Sta-def env-tran-right-def* **by** *blast*
also have *snd* $(l!Suc\ j) \in$ *Normal ‘ q* **using** *Suc*(*8*) **by** *auto*
ultimately show *?thesis* **using** *Suc*(*8*) *l* **by** (*metis env-tran-tail* j' *nth-Cons-Suc*)

qed
ultimately show *?case* **using** *Suc*(*1*)[*of l1 i' j'*] *Suc*(*7*) *Suc*(*8*) j' *l* **by** *auto*
qed

lemma *exists-first-occ*: $P\ (n::nat) \Longrightarrow \exists m.\ P\ m \wedge (\forall i<m.\ \neg\ P\ i)$
proof (*induct n*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc n*) **thus** *?case*
by (*metis ex-least-nat-le not-less0*)
qed

lemma *exist-first-comp-tran'*:
assumes *a1*: $Suc\ i<length\ l \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$
shows $\exists j.\ (Suc\ j<length\ l \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))) \wedge (\forall k < j.\ \neg\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))$
proof −
let $?P = (\lambda n.\ Suc\ n<length\ l \wedge (\Gamma\vdash_c(l!n) \rightarrow (l!(Suc\ n))))$
show *?thesis* **using** *exists-first-occ*[*of ?P i*] *a1* **by** *auto*
qed

lemma *exist-first-comp-tran*:
assumes *a0*:$(\Gamma, l) \in cptn$ **and**
a1: $Suc\ i<length\ l \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$

shows $\exists j.\ j \leq i \wedge\ (\Gamma \vdash_c (l!j)\ \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma \vdash_c (l!k)\ \rightarrow_e (l!(Suc\ k))))$
proof –
  obtain j **where** $pj{:}(Suc\ j{<}length\ l \wedge\ (\Gamma \vdash_c (l!j)\ \rightarrow (l!(Suc\ j)))) \wedge$
      $(\forall k < j.\ \neg(Suc\ k{<}length\ l \wedge\ (\Gamma \vdash_c (l!k)\ \rightarrow (l!(Suc\ k)))))$
    using *a1 exist-first-comp-tran′* **by** *blast*
  then have $j \leq i$ **using** *a1 pj* **by** (*cases* $j \leq i$, *auto*)
  moreover have $\Gamma \vdash_c (l!j)\ \rightarrow (l!(Suc\ j))$ **using** *pj* **by** *auto*
  moreover have $(\forall k < j.\ (\Gamma \vdash_c (l!k)\ \rightarrow_e (l!(Suc\ k))))$
  proof –
    {**fix** k
    assume $kj{:}k{<}j$
    then have $Suc\ k \geq length\ l \vee\ \neg\ (\ (\Gamma \vdash_c (l!k)\ \rightarrow (l!(Suc\ k))))$ **using** *pj* **by** *auto*
    then have $(\Gamma \vdash_c (l!k)\ \rightarrow_e (l!(Suc\ k)))$
    proof
      {**assume** $length\ l \leq Suc\ k$
      thus *?thesis* **using** *kj pj* **by** *auto*
      }
      {**assume** $\neg\ (\Gamma \vdash_c (l!k)\ \rightarrow (l!(Suc\ k)))$
      also have $k + 1 < length\ l$ **using** *kj pj* **by** *auto*
      ultimately show *?thesis*
        using *a0 cptn-tran-ce-i step-ce-elim-cases* **by** *blast*
      }
    qed
    } **thus** *?thesis* **by** *auto*
  qed
  ultimately show *?thesis* **by** *auto*
qed

lemma *skip-com-all-skip*:
assumes $a0{:}(\Gamma,\ l) \in cptn$ **and**
    $a1{:}fst\ (l!i) = Skip$ **and**
    $a2{:}i{<}length\ l$
  shows $\forall j.\ j \geq i \wedge j\ {<}length\ l \longrightarrow fst\ (l!j) = Skip$
using *a0 a1 a2*
proof (*induct length* $l - (i + 1)$ *arbitrary*: i)
 case 0 **thus** *?case* **by** (*metis Suc-eq-plus1 Suc-leI diff-is-0-eq nat-less-le zero-less-diff*)

next
  case $(Suc\ n)$
  then have $l{:}Suc\ i < length\ l$ **by** *arith*
  have $n{:}n = (length\ l) - (Suc\ i + 1)$ **using** *Suc* **by** *arith*
  then have $\Gamma \vdash_c\ l\ !\ i \rightarrow_{ce}\ l\ !\ Suc\ i$ **using** *cptn-tran-ce-i Suc*
   by (*metis* (*no-types*) *Suc.hyps*(*2*) *a0 cptn-tran-ce-i zero-less-Suc zero-less-diff*)
  then have $\Gamma \vdash_c\ l\ !\ i \rightarrow l\ !\ Suc\ i \vee \Gamma \vdash_c\ l\ !\ i \rightarrow_e\ l\ !\ Suc\ i$
    using *step-ce-elim-cases* **by** *blast*
  then have $or{:}fst(l!Suc\ i) = Skip$
  proof

{**assume** $\Gamma\vdash_c l\ !\ i \rightarrow_e l\ !\ Suc\ i$
thus *?thesis* **using** *Suc*(*4*) **by** (*metis env-c-c′ prod.collapse*)
}
next
{**assume** *step*:$\Gamma\vdash_c l\ !\ i \rightarrow l\ !\ Suc\ i$
{**assume** *fst*(*l*!*i*) = *Skip*
then have *?thesis* **using** *step*
using *SmallStepCon.final-def SmallStepCon.no-step-final′* **by** *blast*
}**note** *left* = *this*
{**assume** *fst*(*l*!*i*) = *Throw*
then have *?thesis* **using** *step stepc-elim-cases*
proof −
have $\exists x.\ l\ !\ Suc\ i$ = (*LanguageCon.com.Skip*, *x*)
by (*metis* (*no-types*) ⟨*fst* (*l* ! *i*) = *LanguageCon.com.Throw*⟩ *local.step stepc-elim-cases*(*11*) *surjective-pairing*)
then show *?thesis*
by *fastforce*
qed
} **then show** *?thesis* **using** *Suc*(*4*) *left* **by** *auto*
}
qed
show *?case* **using** *Suc*(*1*)[*OF n a0 or l*] *Suc*(*4*) *Suc*(*5*) **by** (*metis le-less-Suc-eq not-le*)
qed

lemma *terminal-com-all-term*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1:*fst* (*l*!*i*) = *Skip* ∨ *fst* (*l*!*i*) = *Throw* **and**
a2:*i*<*length l*
shows ∀ *j*. *j*≥*i* ∧ *j* <*length l* ⟶ *fst* (*l*!*j*) = *Skip* ∨ *fst* (*l*!*j*) = *Throw*
using *a0 a1 a2*
proof (*induct length l* − (*i* + *1*) *arbitrary*: *i*)
case *0* **thus** *?case* **by** (*metis Suc-eq-plus1 Suc-leI diff-is-0-eq nat-less-le zero-less-diff*)

next
case (*Suc n*)
then have *l*:*Suc i* < *length l* **by** *arith*
have *n*:*n* = (*length l*) − (*Suc i* + *1*) **using** *Suc* **by** *arith*
then have $\Gamma\vdash_c l\ !\ i \rightarrow_{ce} l\ !\ Suc\ i$ **using** *cptn-tran-ce-i Suc*
by (*metis* (*no-types*) *Suc.hyps*(*2*) *a0 cptn-tran-ce-i zero-less-Suc zero-less-diff*)
then have $\Gamma\vdash_c l\ !\ i \rightarrow l\ !\ Suc\ i \lor \Gamma\vdash_c l\ !\ i \rightarrow_e l\ !\ Suc\ i$
using *step-ce-elim-cases* **by** *blast*
then have *or*:*fst*(*l*!*Suc i*) = *Skip* ∨ *fst*(*l*!*Suc i*) = *Throw*
proof
{**assume** $\Gamma\vdash_c l\ !\ i \rightarrow_e l\ !\ Suc\ i$
thus *?thesis* **using** *Suc*(*4*) **by** (*metis env-c-c′ prod.collapse*)
}
next
{**assume** *step*:$\Gamma\vdash_c l\ !\ i \rightarrow l\ !\ Suc\ i$

{**assume** *fst*($l!i$) = *Skip*
then have *?thesis* **using** *step*
using *SmallStepCon.final-def SmallStepCon.no-step-final'* **by** *blast*
}**note** *left* = *this*
{**assume** *fst*($l!i$) = *Throw*
then have *?thesis* **using** *step stepc-elim-cases*
proof −
have $\exists x.\ l\ !\ Suc\ i = (LanguageCon.com.Skip,\ x)$
by (*metis* (*no-types*) ‹*fst* (l ! i) = *LanguageCon.com.Throw*› *local.step stepc-elim-cases*(*11*) *surjective-pairing*)
then show *?thesis*
by *fastforce*
qed
} **then show** *?thesis* **using** *Suc*(*4*) *left* **by** *auto*
}
qed
show *?case* **using** *Suc*(*1*)[*OF n a0 or l*] *Suc*(*4*) *Suc*(*5*) **by** (*metis le-less-Suc-eq not-le*)
qed

lemma *only-one-c-comp-tran*:
assumes *a0*:(Γ, l) $\in$ *cptn* **and**
a1: *fst* ($l!0$) = c **and**
a2: $Suc\ i<length\ l \wedge (\Gamma\vdash_c(l!i) \ \rightarrow (l!(Suc\ i)))$ **and**
a3: $i < j \wedge Suc\ j < length\ l \wedge (\Gamma\vdash_c(l!j) \ \rightarrow (l!(Suc\ j))) \wedge fst\ (l!j) = c$ **and**
a4: $\forall s1\ s2\ c1.\ \Gamma\vdash_c(c,\ s1) \ \rightarrow ((c1,s2)) \longrightarrow$
$((c1=Skip) \vee (c1=Throw))$ **and**
a5: $(\forall k < i.\ (\Gamma\vdash_c(l!k) \ \rightarrow_e (l!(Suc\ k))))$
shows P
proof −
have *fst*:*fst* ($l!i$) = c **using** *a0 a1 a5*
by (*simp add*: *a2 cptn-env-same-prog*)
then have *suci*:*fst* ($l!Suc\ i$) = *Skip* $\vee$ *fst* ($l!Suc\ i$) = *Throw*
using *a4* **by** (*metis a2 surjective-pairing*)
then have *fst* ($l!j$) = *Skip* $\vee$ *fst* ($l!j$) = *Throw*
proof −
have $Suc\ i \leq j$
using *Suc-leI a3* **by** *presburger*
then show *?thesis*
using *Suc-lessD terminal-com-all-term*[*OF a0 suci*] *a2 a3* **by** *blast*
qed
thus *?thesis*
proof
{**assume** *fst* (l ! j) = *Skip*
then show *?thesis* **using** *a3 SmallStepCon.final-def SmallStepCon.no-step-final'* **by** *blast*
}
next

{**assume** *asm*:*fst* $(l \,!\, j) = $ *Throw*
then show *?thesis*
proof (*cases snd* (*l*!*i*))
case *Normal*
thus *?thesis* **using** *a3 a2 fst asm*
by (*metis SmallStepCon.final-def SmallStepCon.no-step-final′*)
next
case *Abrupt* **thus** *?thesis* **using** *a3 a2 fst asm skip-com-all-skip*
suci **by** (*metis Suc-leI Suc-lessD a0 mod-env-not-component prod.collapse*)

next
case *Fault* **thus** *?thesis* **using** *a3 a2 fst asm skip-com-all-skip*
suci **by** (*metis Suc-leI Suc-lessD a0 mod-env-not-component prod.collapse*)
next
case *Stuck* **thus** *?thesis* **using** *a3 a2 fst asm skip-com-all-skip*
suci **by** (*metis Suc-leI Suc-lessD a0 mod-env-not-component prod.collapse*)
qed
}
qed
qed

lemma *only-one-component-tran1*:
assumes *a0*:$(\Gamma, l) \in$ *cptn* **and**
a1: $fst\ (l!0) = c$ **and**
a2: $Suc\ i < length\ l \wedge (\Gamma\vdash_c (l!i) \ \rightarrow (l!(Suc\ i)))$ **and**
a3: $j \neq i \wedge Suc\ j < length\ l \wedge (\Gamma\vdash_c (l!j) \ \rightarrow (l!(Suc\ j))) \wedge fst\ (l!j) = c$
and
a4: $\forall s1\ s2\ c1.\ \Gamma\vdash_c (c,\ s1) \ \rightarrow ((c1,s2)) \longrightarrow$
$((c1{=}Skip) \vee (c1{=}Throw))$ **and**
a5: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!0) \in Normal\ `\ p \wedge$
$Sta\ q\ rely \wedge snd\ (l!Suc\ j) \in Normal\ `\ q$
shows *P*
proof (*cases j=i*)
case *True* **thus** *?thesis* **using** *a3* **by** *auto*
next
case *False* **note** *j-neq-i=this*
thus *?thesis*
proof (*cases j<i*)
case *True*
thus *?thesis*
proof −
obtain $bb :: {'b}\ set \Rightarrow ({'b} \Rightarrow ({'b},\ {'c})\ xstate) \Rightarrow ({'b},\ {'c})\ xstate \Rightarrow {'b}$ **where**
$\forall x0\ x1\ x2.\ (\exists v3.\ x2 = x1\ v3 \wedge v3 \in x0) = (x2 = x1\ (bb\ x0\ x1\ x2) \wedge bb\ x0\ x1\ x2 \in x0)$
by *moura*
then have *f1*: $\forall x\, f\, B.\ x \notin f\ `\ B \vee x = f\ (bb\ B\, f\, x) \wedge bb\ B\, f\, x \in B$
by (*meson imageE*)
then have $\Gamma\vdash_c (c,\ snd\ (l \,!\, j)) \rightarrow (fst\ (l \,!\, Suc\ j),\ Normal\ (bb\ q\ Normal\ (snd\ (l \,!\, Suc\ j))))$

```
      by (metis (no-types) a3 a5 surjective-pairing)
    then show ?thesis
      using f1 by (meson Suc-leI a0 a2 a4 a5 True only-one-component-tran-j)
  qed
next
  case False
  obtain j1
  where all-ev:j1≤i ∧
          (Γ⊢c(l!j1) → (l!(Suc j1))) ∧
          (∀ k < j1. (Γ⊢c(l!k) →e (l!(Suc k))))
    using a0 a2 a3 exist-first-comp-tran by blast
  then have fst:fst (l!j1) = c
    using a0 a1 a2 cptn-env-same-prog le-imp-less-Suc less-trans-Suc by blast
  have suc:Suc j1 < length l ∧ Γ⊢c l ! j1 → l ! Suc j1 using all-ev a2
     using Suc-lessD le-eq-less-or-eq less-trans-Suc by linarith
  have evs:(∀ k < j1. (Γ⊢c(l!k) →e (l!(Suc k)))) using all-ev by auto
  have j:j1 < j ∧ Suc j < length l ∧ Γ⊢c l ! j → l ! Suc j ∧ fst (l ! j) = c
    using a3 all-ev False by auto
  then show ?thesis
    using only-one-c-comp-tran[OF a0 a1 suc j a4 evs] by auto
qed
qed

lemma only-one-component-tran-i:
 assumes a0:(Γ, l) ∈ cptn and
       a1: fst (l!k) = c and
       a2: Suc i<length l ∧ k≤i ∧ (Γ⊢c(l!i) → (l!(Suc i))) and
       a3: k≤j ∧ j ≠ i ∧ Suc j < length l ∧ (Γ⊢c(l!j) → (l!(Suc j))) ∧ fst (l!j)
= c and
       a4: ∀ s1 s2 c1. Γ⊢c(c, s1) → ((c1,s2)) ⟶
              ((c1=Skip) ∨ (c1=Throw)) and
       a5: env-tran-right Γ l rely ∧ Sta p rely ∧ snd (l!k) ∈ Normal ‘ p ∧
                                Sta q rely ∧ snd (l!Suc j) ∈ Normal ‘ q
  shows P
using a0 a1 a2 a3 a4 a5
proof (induct k arbitrary: l i j p q)
  case 0 show ?thesis using only-one-component-tran1[OF 0(1) 0(2) ] 0 by
blast
next
 case (Suc n)
  then obtain a1 l1 where l: l=a1#l1
   by (metis less-nat-zero-code list.exhaust list.size(3))
 then have l1notempty:l1≠[] using Suc by force
 then obtain i' where i': i=Suc i' using Suc
   using less-imp-Suc-add using Suc-le-D by meson
 then obtain j' where j': j=Suc j' using Suc
   using Suc-le-D by meson
 have a0:(Γ,l1)∈cptn using Suc l
   using tl-in-cptn l1notempty by meson
```

moreover have *a1*:*fst* (*l1* ! *n*) = *c*
using *Suc l l1notempty* **by** *force*
moreover have *a2*:*Suc i'* < *length l1* ∧ *n* ≤ *i'* ∧ $\Gamma\vdash_c$ *l1* ! *i'* → (*l1* ! *Suc i'*)
using *Suc l l1notempty i'* **by** *auto*
moreover have *a3*:*n* ≤ *j'* ∧ *j'* ≠ *i'* ∧ *Suc j'* < *length l1* ∧ $\Gamma\vdash_c$ *l1* ! *j'* → (*l1* ! *Suc j'*) ∧ *fst* (*l1*!*j'*) = *c*
using *Suc l l1notempty i' j'* **by** *auto*
moreover have *a4*:*env-tran-right* Γ *l1 rely* ∧
Sta p rely ∧ *snd* (*l1*!*n*) ∈ *Normal* ' *p* ∧
Sta q rely ∧ *snd* (*l1* ! *Suc j'*) ∈ *Normal* ' *q*
using *Suc*(*7*) *l j'* **unfolding** *env-tran-right-def* **by** *fastforce*
show *?case* **using** *Suc*(*1*)[*OF a0 a1 a2 a3 Suc*(*6*) *a4*] **by** *auto*
qed

lemma *only-one-component-tran*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*k*) = *c* **and**
a2: *k*≤*i* ∧ *i* ≠ *j* ∧ *Suc i*<*length l* ∧ ($\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*))) ∧ *fst* (*l*!*i*) = *c* **and**
a3: *k*≤*j* ∧ *Suc j* < *length l* **and**
a4: ∀ *s1 s2 c1*. $\Gamma\vdash_c$(*c*,*s1*) → ((*c1*,*s2*)) ⟶
((*c1*=*Skip*) ∨ (*c1*=*Throw*)) **and**
a5: *env-tran-right* Γ *l rely* ∧ *Sta p rely* ∧ *snd* (*l*!*k*) ∈ *Normal* ' *p* ∧
Sta q rely ∧ *snd* (*l*!*Suc i*) ∈ *Normal* ' *q*
shows ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))
using *a0 a1 a2 a3 a4 a5 only-one-component-tran-i*
proof −
{**assume** ($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*))) ∨ (¬ $\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*)))
then have ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))
proof
assume $\Gamma\vdash_c$ *l* ! *j* → (*l* ! *Suc j*)
then have *j*:*Suc j*<*length l* ∧ *k*≤*j* ∧ ($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*))) **using** *a3* **by** *auto*
show *?thesis* **using** *only-one-component-tran-i*[*OF a0 a1 j a2 a4 a5*]
by *blast*
next
assume ¬ $\Gamma\vdash_c$ *l* ! *j* → (*l* ! *Suc j*)
thus *?thesis*
by (*metis Suc-eq-plus1 a0 a3 cptn-tran-ce-i step-ce-elim-cases*)
qed
} **thus** *?thesis* **by** *auto*
qed

lemma *only-one-component-tran-all-env*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*k*) = *c* **and**
a2: *Suc i*<*length l* ∧ *k*≤*i* ∧ ($\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*))) ∧ *fst* (*l*!*i*) = *c* **and**
a3: ∀ *s1 s2 c1*. $\Gamma\vdash_c$(*c*,*s1*) → ((*c1*,*s2*)) ⟶
((*c1*=*Skip*) ∨ (*c1*=*Throw*)) **and**

a4: *env-tran-right* Γ *l rely* ∧ *Sta p rely* ∧ *snd* (*l*!*k*) ∈ *Normal* ‘ *p* ∧
Sta q rely ∧ *snd* (*l*!*Suc i*) ∈ *Normal* ‘ *q*
shows ∀ *j*. *k*≤*j* ∧ *j*≠*i* ∧ *Suc j* < (*length l*) ⟶ (Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*)))
proof −
{**fix** *j*
assume *ass*:*k*≤*j* ∧ *j*≠*i* ∧ *Suc j* < (*length l*)
then have *a2*:*k* ≤ *i* ∧ *i* ≠ *j* ∧ *Suc i* < *length l* ∧ Γ⊢$_c$ *l* ! *i* → *l* ! *Suc i* ∧ *fst* (*l* ! *i*) = *c*
using *a2* **by** *auto*
then have (Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*)))
using *only-one-component-tran*[*OF a0 a1*] *a2 a3 ass a4* **by** *blast*
} **thus** *?thesis* **by** *auto*
qed

lemma *only-one-component-tran-all-not-comp*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*k*) = *c* **and**
a2: *Suc i*<*length l* ∧ *k*≤*i* ∧ (Γ⊢$_c$(*l*!*i*) → (*l*!(*Suc i*))) ∧ *fst* (*l*!*i*) = *c* **and**
a3: ∀ *s1 s2 c1*. Γ⊢$_c$(*c*, *s1*) → ((*c1*,*s2*)) ⟶
((*c1*=*Skip*) ∨ (*c1*=*Throw*)) **and**
a4: *env-tran-right* Γ *l rely* ∧ *Sta p rely* ∧ *snd* (*l*!*k*) ∈ *Normal* ‘ *p* ∧
Sta q rely ∧ *snd* (*l*!*Suc i*) ∈ *Normal* ‘ *q*
shows ∀ *j*. *k*≤*j* ∧ *j*≠*i* ∧ *Suc j* < (*length l*) ⟶ ¬(Γ⊢$_c$(*l*!*j*) → (*l*!(*Suc j*)))
proof −
{**fix** *j*
assume *ass*:*k*≤*j* ∧ *j*≠*i* ∧ *Suc j* < (*length l*)
then have ¬(Γ⊢$_c$(*l*!*j*) → (*l*!(*Suc j*)))
using *a0 a1 a2 a3 a4 only-one-component-tran-i ass* **by** *blast*
} **thus** *?thesis* **by** *auto*
qed

lemma *final-exist-component-tran1*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*i*) = *c* **and**
a2: *env-tran* Γ *q l R* ∧ *Sta q R* **and**
a3: *i*≤*j* ∧ *j* < *length l* ∧ *final* (*l*!*j*) **and**
a5: *c*≠*Skip* ∧ *c*≠*Throw*
shows ∃ *k*. *k*≥*i* ∧ *k*<*j* ∧ (Γ⊢$_c$(*l*!*k*) → (*l*!(*Suc k*)))
proof −
{**assume** ∀ *k*. *k*≥*i* ∧ *k*<*j* ⟶ ¬(Γ⊢$_c$(*l*!*k*) → (*l*!(*Suc k*)))
then have ∀ *k*. *k*≥*i* ∧ *k*<*j* ⟶ (Γ⊢$_c$(*l*!*k*) →$_e$ (*l*!(*Suc k*)))
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 a0 a3 cptn-tran-ce-i less-trans-Suc step-ce-elim-cases*)
then have *fst* (*l*!*j*) = *fst* (*l*!*i*) **using** *cptn-i-env-same-prog a0 a3* **by** *blast*
then have *False* **using** *a3 a1 a5* **unfolding** *final-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *final-exist-component-tran*:
  assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
    a1: *fst* (*l*!*i*) = *c* **and**
    a2: *i*$\leq$*j* $\wedge$ *j* < *length l* $\wedge$ *final* (*l*!*j*) **and**
    a3: *c*$\neq$*Skip* $\wedge$ *c*$\neq$*Throw*
  shows $\exists$ *k*. *k*$\geq$*i* $\wedge$ *k*<*j* $\wedge$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*)))
proof −
  {**assume** $\forall$ *k*. *k*$\geq$*i* $\wedge$ *k*<*j* $\longrightarrow$ $\neg$($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*)))
   then have $\forall$ *k*. *k*$\geq$*i* $\wedge$ *k*<*j* $\longrightarrow$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow_e$ (*l*!(*Suc k*)))
    by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 a0 a2 cptn-tran-ce-i less-trans-Suc step-ce-elim-cases*)
   then have *fst* (*l*!*j*) = *fst* (*l*!*i*) **using** *cptn-i-env-same-prog a0 a2* **by** *blast*
   then have *False* **using** *a2 a1 a3* **unfolding** *final-def* **by** *auto*
  }
  thus *?thesis* **by** *auto*
qed

lemma *suc-not-final-final-c-tran*:
 assumes *a0*: (Γ, *l*) $\in$ *cptn* **and**
   a1: *Suc j*< *length l* $\wedge$ $\neg$*final* (*l*!*j*) $\wedge$ *final* (*l*!*Suc j*)
 shows ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow$ (*l*!(*Suc j*)))
proof −
   obtain *x xs* **where** *l*:*l* = *x*#*xs* **using** *a0 cptn.simps* **by** *blast*
   obtain *c1 s1 c2 s2* **where** *l1*:*l*!*j* = (*c1*,*s1*) $\wedge$ *l*!(*Suc j*) = (*c2*,*s2*) **using** *a1* **by** *fastforce*
   have $\neg$ $\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*))
   proof −
     { **assume** *a*:$\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*))
       then have *eq-fst*:*fst* (*l*!*j*) = *fst* (*l*!*Suc j*) **by** (*metis env-c-c′ prod.collapse*)
       { **assume** *fst* (*l*!*Suc j*) = *Skip*
         then have *False* **using** *a1 eq-fst* **unfolding** *final-def* **by** *fastforce*
       }**note** *p1*=*this*
       { **assume** *fst* (*l*!*Suc j*) = *Throw* $\wedge$ ($\exists$ *s*. *snd* (*l*!*Suc j*) = *Normal s*)
         then have *False* **using** *a1 eq-fst* **unfolding** *final-def*
         by (*metis a eenv-normal-s′-normal-s local.l1 snd-conv*)
       }
       then have *False* **using** *a1 p1* **unfolding** *final-def* **by** *fastforce*
     } **thus** *?thesis* **by** *auto*
   qed
   also have $\Gamma\vdash_c$(*l*!*j*) $\rightarrow_{ce}$ (*l*!(*Suc j*)) **using** *l cptn-stepc-rtran a0 a1* **by** *fastforce*

   ultimately show *?thesis* **using** *step-ce-not-step-e-step-c local.l1* **by** *fastforce*
qed

lemma *final-exist-component-tran-final*:
  assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
    a2: *i*$\leq$*j* $\wedge$ *j* < *length l* $\wedge$ *final* (*l*!*j*) **and**
    a3: $\neg$*final*(*l*!*i*)
  shows $\exists$ *k*. *k*$\geq$*i* $\wedge$ *k*<*j* $\wedge$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*))) $\wedge$ *final*(*l*!(*Suc k*))

proof −
  let $?P = \lambda j.\ i \leq j \wedge j < length\ l \wedge final\ (l!j)$
  obtain k **where** $k{:}?P\ k \wedge (\forall i<k.\ \neg\ ?P\ i)$ **using** *a2 exists-first-occ*[*of ?P j*] **by** *auto*
  then have $i\text{-}k\text{-}not\text{-}final{:}\forall i'<k.\ i' \geq i \longrightarrow \neg final\ (l!i')$ **using** *a2* **by** *fastforce*
  have $i\text{-}eq\text{-}j{:}i<j$ **using** *a2 a3* **using** *le-imp-less-or-eq* **by** *auto*
  then obtain *pre-k* **where** $pre\text{-}k{:}Suc\ pre\text{-}k = k$ **using** *a2 k*
    by (*metis a3 eq-iff le0 lessE neq0-conv*)
  then have $\Gamma\vdash_c(l!pre\text{-}k) \rightarrow (l!k)$
  proof −
    have $pre\text{-}k \geq i$ **using** *pre-k i-eq-j* **using** *a3 k le-Suc-eq* **by** *blast*
    then have $\neg(final\ (l!pre\text{-}k))$ **using** *i-k-not-final pre-k* **by** *auto*
    thus *?thesis* **using** *suc-not-final-final-c-tran a0 a2 pre-k k* **by** *fastforce*
  qed
  thus *?thesis* **using** *pre-k* **by** (*metis a2 a3 i-k-not-final k le-Suc-eq not-less-eq*)
qed

12.2 Basic Sound

lemma *basic-skip*:
  $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Basic\ f\ e,\ s1) \rightarrow ((c1,s2)) \longrightarrow c1=Skip$
proof −
  {**fix** *s1 s2 c1*
   assume $\Gamma\vdash_c(Basic\ f\ e,s1) \rightarrow ((c1,s2))$
   then have $c1=Skip$ **using** *stepc-elim-cases*(*3*) **by** *blast*
  } **thus** *?thesis* **by** *auto*
qed

lemma *no-comp-tran-before-i-basic*:
  assumes $a0{:}(\Gamma,\ l) \in cptn$ **and**
      $a1$: $fst\ (l!k) = Basic\ f\ e$ **and**
      $a2$: $Suc\ i<length\ l \wedge k\leq i \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$ **and**
      $a3$: $k\leq j \wedge j < i \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))$ **and**
      $a4$: $\forall k < j.\ (\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k)))$ **and**
      $a5$: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!0) \in Normal\ `\ p\ \wedge$
            $Sta\ q\ rely \wedge snd\ (l!Suc\ j) \in Normal\ `\ q$
   shows P
proof −
  have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Basic\ f\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
    using *basic-skip* **by** *fastforce*
  thus *?thesis* **using** *a0 a1 a2 a3 a4 a5 no-comp-tran-before-i* **by** *blast*
qed

lemma *only-one-component-tran-i-basic*:
  assumes $a0{:}(\Gamma,\ l) \in cptn$ **and**
      $a1$: $fst\ (l!k) = Basic\ f\ e$ **and**
      $a2$: $Suc\ i<length\ l \wedge k\leq i \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$ **and**
      $a3$: $k\leq j \wedge j \neq i \wedge Suc\ j < length\ l \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j))) \wedge fst\ (l!j) = Basic\ f\ e$ **and**

$a4$: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!k) \in$ *Normal* ‘ p $\wedge$
Sta q rely $\wedge$ *snd* $(l!Suc\ j) \in$ *Normal* ‘ q
shows P
proof –
have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c(Basic\ f\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *basic-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran-i*[*OF a0 a1 a2*] **by** *blast*
qed

lemma *only-one-component-tran-basic*:
assumes *a0*:$(\Gamma, l) \in$ *cptn* **and**
a1: *fst* $(l!k) =$ *Basic f e* **and**
a2: $k\leq i \wedge i \neq j \wedge$ *Suc* $i<$*length* $l \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i))) \wedge$ *fst* $(l!i)$ $=$ *Basic f e* **and**
a3: $k\leq j \wedge$ *Suc* $j <$ *length* l **and**
a4: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!k) \in$ *Normal* ‘ p $\wedge$
Sta q rely $\wedge$ *snd* $(l!Suc\ i) \in$ *Normal* ‘ q
shows $(\Gamma\vdash_c(l!j) \rightarrow_e (l!(Suc\ j)))$
proof –
have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c(Basic\ f\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *basic-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran* **by** *blast*
qed

lemma *only-one-component-tran-all-env-basic*:
assumes *a0*:$(\Gamma, l) \in$ *cptn* **and**
a1: *fst* $(l!k) =$ *Basic f e* **and**
a2: $k\leq i \wedge$ *Suc* $i<$*length* $l \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i))) \wedge$ *fst* $(l!i) =$ *Basic f e* **and**
a3: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!k) \in$ *Normal* ‘ p $\wedge$
Sta q rely $\wedge$ *snd* $(l!Suc\ i) \in$ *Normal* ‘ q
shows $\forall j.\ k\leq j \wedge j\neq i \wedge$ *Suc* $j <$ (*length* l) $\longrightarrow (\Gamma\vdash_c(l!j) \rightarrow_e (l!(Suc\ j)))$
proof –
have *b*: $\forall$ *s1 s2 c1*. $\Gamma\vdash_c(Basic\ f\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *basic-skip* **by** *blast*
show *?thesis*
by (*metis* (*no-types*) *a0 a1 a2 a3 only-one-component-tran-basic*)
qed

lemma *only-one-component-tran-all-not-comp-basic*:
assumes *a0*:$(\Gamma, l) \in$ *cptn* **and**
a1: *fst* $(l!k) =$ *Basic f e* **and**
a2: *Suc* $i<$*length* $l \wedge k\leq i \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i))) \wedge$ *fst* $(l!i) =$ *Basic f e* **and**
a3: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!k) \in$ *Normal* ‘ p $\wedge$
Sta q rely $\wedge$ *snd* $(l!Suc\ i) \in$ *Normal* ‘ q
shows $\forall j.\ k\leq j \wedge j\neq i \wedge$ *Suc* $j <$ (*length* l) $\longrightarrow \neg(\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))$
proof –

have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c$(*Basic f e*,*s1*) $\rightarrow$ ((*c1*,*s2*)) $\longrightarrow$ (*c1*=*Skip*)
using *basic-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 only-one-component-tran-all-not-comp* **by** *blast*
qed

lemma *one-component-tran-basic*:
assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
a1: *fst* (*l*!*0*) = *Basic f e* **and**
a2: *Suc k*<*length l* $\wedge$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*))) **and**
a3: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*0*) $\in$ *Normal* ' *p* $\wedge$
Sta q rely **and**
a4:*p* $\subseteq$ {*s*. *f s* $\in$ *q*}

shows $\forall$ *j*. *0*$\leq$*j* $\wedge$ *j*$\neq$*k* $\wedge$ *Suc j* < (*length l*) $\longrightarrow$ $\neg$($\Gamma\vdash_c$(*l*!*j*) $\rightarrow$ (*l*!(*Suc j*)))
proof –
have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c$(*Basic f e*,*s1*) $\rightarrow$ ((*c1*,*s2*)) $\longrightarrow$ (*c1*=*Skip*)
using *basic-skip* **by** *blast*
also obtain *j* **where** *first*:(*Suc j*<*length l* $\wedge$ ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow$ (*l*!(*Suc j*)))) $\wedge$
($\forall$ *k* < *j*. $\neg$(($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*)))))
by (*metis* (*no-types*) *a2 exist-first-comp-tran'*)
moreover then have *prg-j*:*fst* (*l*!*j*) = *Basic f e* **using** *a1 a0*
by (*metis cptn-env-same-prog not-step-comp-step-env*)
moreover have *sta-j*:*snd* (*l*!*j*) $\in$ *Normal* ' *p*
proof –
have *a0'*:*0*$\leq$*j* $\wedge$ *j*<(*length l*) **using** *first* **by** *auto*
have *a1'*:($\forall$ *k*. *0*$\leq$*k* $\wedge$ *k* < *j* $\longrightarrow$ (($\Gamma\vdash_c$(*l*!*k*) $\rightarrow_e$ (*l*!(*Suc k*)))))
using *first not-step-comp-step-env a0* **by** *fastforce*
thus *?thesis* **using** *stability first a3 a1' a0'* **by** *blast*
qed
then have *snd* (*l*!*Suc j*) $\in$ *Normal* ' *q* **using** *a4 first prg-j*
proof –
obtain *s* **where** *snd* (*l*!*j*) = *Normal s* $\wedge$ *s*$\in$*p* **using** *sta-j* **by** *fastforce*
moreover then have *fst*(*l*!*Suc j*) = *Skip* $\wedge$ *snd*(*l*!*Suc j*) = *Normal* (*f s*) **using** *first*
by (*metis fst-conv prg-j snd-conv stepc-Normal-elim-cases*(*3*) *surjective-pairing*)
ultimately show *?thesis* **using** *a4* **by** *fastforce*
qed
then have $\forall$ *i*. *0*$\leq$*i* $\wedge$ *i*$\neq$*j* $\wedge$ *Suc i* < (*length l*) $\longrightarrow$ $\neg$($\Gamma\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*)))
using *only-one-component-tran-all-not-comp-basic*[*OF a0 a1*] *first a3*
a0 a1 calculation(*1*) *only-one-component-tran1 prg-j* **by** *blast*
moreover then have *k*=*j* **using** *a2* **by** *fastforce*
ultimately show *?thesis* **by** *auto*
qed

lemma *one-component-tran-basic-env*:
assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
a1: *fst* (*l*!*0*) = *Basic f e* **and**
a2: *Suc k*<*length l* $\wedge$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*))) **and**
a3: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*0*) $\in$ *Normal* ' *p* $\wedge$

Sta q rely **and**
a4:*p* ⊆ {*s*. *f s* ∈ *q*}
shows ∀ *j*. *0*≤*j* ∧ *j*≠*k* ∧ *Suc j* < (*length l*) ⟶ Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*))
proof −
have ∀ *j*. *0*≤*j* ∧ *j*≠*k* ∧ *Suc j* < (*length l*) ⟶ ¬ (Γ⊢$_c$(*l*!*j*) → (*l*!(*Suc j*)))
using *one-component-tran-basic*[*OF a0 a1 a2 a3 a4*] **by** *auto*
thus *?thesis* **using** *a0*
by (*metis Suc-eq-plus1 cptn-tran-ce-i step-ce-elim-cases*)
qed

lemma *final-exist-component-tran-basic*:
assumes *a0*:(Γ, *l*) ∈ *cptn* **and**
a1: *fst* (*l*!*i*) = *Basic f e* **and**
a2: *env-tran* Γ *q l R* **and**
a3: *i*≤*j* ∧ *j* < *length l* ∧ *final* (*l*!*j*)
shows ∃ *k*. *k*≥*i* ∧ *k*<*j* ∧ (Γ⊢$_c$(*l*!*k*) → (*l*!(*Suc k*)))
proof −
show *?thesis* **using** *a0 a1 a2 a3 final-exist-component-tran* **by** *blast*
qed

lemma *Basic-sound1*:
assumes *a0*:*p* ⊆ {*s*. *f s* ∈ *q*} **and**
a1:(∀ *s t*. *s*∈*p* ∧ (*t*=*f s*) ⟶ (*Normal s*,*Normal t*) ∈ *G*) **and**
a2:*Sta p R* **and**
a3:*Sta q R* **and**
a10:*c* ∈ *cp* Γ (*Basic f e*) *s* **and** *a11*:*c* ∈ *assum*(*p*, *R*)
shows *c*∈*comm*(*G*, (*q*,*a*)) *F*
proof −
obtain Γ*1 l* **where** *c-prod*:*c*=(Γ*1*,*l*) **by** *fastforce*
have *cp*:*l*!*0*=(*Basic f e*,*s*) ∧ (Γ,*l*) ∈ *cptn* ∧ Γ=Γ*1* **using** *a10 cp-def c-prod* **by** *fastforce*
have *assum*:*snd*(*l*!*0*) ∈ *Normal* ‘ (*p*) ∧ (∀ *i*. *Suc i*<*length l* ⟶
(Γ*1*)⊢$_c$(*l*!*i*) →$_e$ (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *R*)
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
have *concl*:(∀ *i ns ns'*. *Suc i*<*length l* ⟶
Γ*1*⊢$_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)
proof −
{ **fix** *k*
assume *a00*:*Suc k*<*length l* **and**
a11:Γ*1*⊢$_c$(*l*!*k*) → (*l*!(*Suc k*))
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a l1* **where** *l*:*l*=*a*#*l1* **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a l1*] *l* **by** *fastforce*
have *env-tran*:*env-tran* Γ *p l R* **using** *assum env-tran-def cp* **by** *blast*

then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def a2* **unfolding** *env-tran-def* **by** *auto*
then have *all-event*:$\forall j.\ 0 \le j \wedge j \neq k \wedge Suc\ j < length\ l \longrightarrow (\Gamma\vdash_c(l!j) \ \rightarrow_e (l!(Suc\ j)))$
using *one-component-tran-basic-env*[*of* Γ *l f e k R*] *a0 a00 a11 a2 a3 assum cp*
env-tran-right fst-conv
by *metis*
then have *before-k-all-evn*:$\forall j.\ 0 \le j \wedge j < k \longrightarrow (\Gamma\vdash_c(l!j) \ \rightarrow_e (l!(Suc\ j)))$
using *a00 a11* **by** *fastforce*
then have *k-basic*:$fst(l!k) = Basic\ f\ e \wedge snd\ (l!k) \in Normal\ `\ (p)$
using *cp env-tran-right a2 assum a00 a11 stability*[*of p R l 0 k k* Γ]
by *force*
have *suc-k-skip-q*:$fst(l!Suc\ k) = Skip \wedge snd\ (l!(Suc\ k)) \in Normal\ `\ q$
proof
show *suc-skip*: $fst(l!Suc\ k) = Skip$
using *a0 a00 a11 k-basic* **by** (*metis basic-skip surjective-pairing*)
next
obtain s' **where** *k-s*: $snd\ (l!k)=Normal\ s' \wedge s' \in (p)$
using *a00 a11 k-basic* **by** *auto*
then have $snd\ (l!(Suc\ k)) = Normal\ (f\ s')$
using *a00 a11 k-basic stepc-Normal-elim-cases*(*3*)
by (*metis prod.inject surjective-pairing*)
then show $snd\ (l!(Suc\ k)) \in Normal\ `\ q$ **using** *a0 k-s* **by** *blast*
qed
obtain $s'\ s''$ **where**
ss:$snd\ (l!k) = Normal\ s' \wedge s' \in (p) \wedge$
$snd\ (l!(Suc\ k)) = Normal\ s'' \wedge s'' \in q$
using *suc-k-skip-q k-basic* **by** *fastforce*
then have $(snd(l!k),\ snd(l!(Suc\ k))) \in G$
using *a0 a1 a2*
by (*metis Pair-inject a11 k-basic prod.exhaust-sel stepc-Normal-elim-cases*(*3*))
} **thus** *?thesis* **by** *auto* **qed**
have *concr*:$(final\ (last\ l) \longrightarrow$
$snd\ (last\ l) \notin Fault\ `\ F \longrightarrow$
$((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)))$
proof−
{
assume *valid*:*final* (*last l*)
have *len-l*:$length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a l1* **where** *l*:$l=a\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:$last\ l = l!(length\ l-1)$
using *last-length* [*of a l1*] *l* **by** *fastforce*
have *env-tran*:*env-tran* Γ *p l R* **using** *assum env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def a2* **unfolding** *env-tran-def* **by** *auto*

have $\exists k.\ k \geq 0 \wedge k < ((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \ \rightarrow (l!(Suc\ k)))$
proof $-$
have $0 \leq (length\ l - 1)$ **using** *len-l last-l* **by** *auto*
moreover have $(length\ l - 1) < length\ l$ **using** *len-l* **by** *auto*
moreover have *final* $(l!(length\ l - 1))$ **using** *valid last-l* **by** *auto*
moreover have *fst* $(l!0) = Basic\ f\ e$ **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-basic env-tran a2* **by** *blast*
qed
then obtain k **where** *k-comp-tran*: $k \geq 0 \wedge k < ((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(Suc\ k)))$
by *auto*
moreover then have $Suc\ k < length\ l$ **by** *auto*
ultimately have *all-event*: $\forall j.\ 0 \leq j \wedge j \neq k \wedge Suc\ j < length\ l \longrightarrow (\Gamma \vdash_c (l!j) \rightarrow_e (l!(Suc\ j)))$
using *one-component-tran-basic-env*[*of* Γ *l f e k R*] *a0 a11 a2 a3 assum cp env-tran-right fst-conv* **by** *metis*
then have *before-k-all-evn*: $\forall j.\ 0 \leq j \wedge j < k \ \longrightarrow (\Gamma \vdash_c (l!j) \ \rightarrow_e (l!(Suc\ j)))$
using *k-comp-tran* **by** *fastforce*
then have *k-basic*: $fst(l!k) = Basic\ f\ e \wedge snd\ (l!k) \in Normal\ `\ (p)$
using *cp env-tran-right a2 assum k-comp-tran stability*[*of p R l 0 k k* Γ]
by *force*
have *suc-k-skip-q*: $fst(l!Suc\ k) = Skip \wedge snd\ (l!(Suc\ k)) \in Normal\ `\ q$
proof
show *suc-skip*: $fst(l!Suc\ k) = Skip$
using *a0 k-comp-tran k-basic* **by** (*metis basic-skip surjective-pairing*)
next
obtain s' **where** *k-s*: $snd\ (l!k) = Normal\ s' \wedge s' \in (p)$
using *k-comp-tran k-basic* **by** *auto*
then have $snd\ (l!(Suc\ k)) = Normal\ (f\ s')$
using *k-comp-tran k-basic stepc-Normal-elim-cases*(*3*)
by (*metis prod.inject surjective-pairing*)
then show $snd\ (l!(Suc\ k)) \in Normal\ `\ q$ **using** *a0* **using** *k-s* **by** *blast*
qed
have *after-k-all-evn*: $\forall j.\ (Suc\ k) \leq j \wedge Suc\ j < (length\ l) \ \longrightarrow (\Gamma \vdash_c (l!j) \ \rightarrow_e (l!(Suc\ j)))$
using *all-event k-comp-tran* **by** *fastforce*
then have *fst-last-skip*: $fst\ (last\ l) = Skip\ \wedge$
$snd\ ((last\ l)) \in Normal\ `\ q$
using *a2 last-l len-l cp env-tran-right a3 suc-k-skip-q assum k-comp-tran stability* [*of q R l Suc k* $((length\ l) - 1)$ - Γ]
by *fastforce*
} **thus** *?thesis* **by** *auto* **qed**
note *res* = *conjI* [*OF concl concr*]
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *auto*
qed

lemma *Basic-sound*:

$p \subseteq \{s.\ f\ s \in q\} \Longrightarrow$
$(\forall s\ t.\ s{\in}p \wedge (t{=}f\ s) \longrightarrow (Normal\ s{,}Normal\ t) \in G) \Longrightarrow$
$Sta\ p\ R \Longrightarrow$
$Sta\ q\ R \Longrightarrow$
$\Gamma{,}\Theta \models n_{/F}\ \ (Basic\ f\ e)\ sat\ [p,\ R,\ G,\ q{,}a]$

proof −
assume
a0:$p \subseteq \{s.\ f\ s \in q\}$ **and**
a1:$(\forall s\ t.\ s{\in}p \wedge (t{=}f\ s) \longrightarrow (Normal\ s{,}Normal\ t) \in G)$ **and**
a2:$Sta\ p\ R$ **and**
a3:$Sta\ q\ R$
{
fix s
have $cpn\ n\ \Gamma\ (Basic\ f\ e)\ s \cap assum(p,\ R) \subseteq comm(G,\ (q{,}a))\ F$
proof −
{
fix c
assume *a10*:$c \in cpn\ n\ \Gamma\ (Basic\ f\ e)\ s$ **and** *a11*:$c \in assum(p,\ R)$
then have *a10*:$c{\in}cp\ \Gamma\ (Basic\ f\ e)\ s$
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
have $c{\in}comm(G,\ (q{,}a))\ F$ **using** *Basic-sound1*[*OF a0 a1 a2 a3 a10 a11*] **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

12.3 Spec Sound

lemma *spec-skip*:
$\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e{,}s1)\ \rightarrow ((c1{,}s2)) \longrightarrow c1{=}Skip$
proof −
{**fix** *s1 s2 c1*
assume $\Gamma\vdash_c(Spec\ r\ e{,}s1)\ \rightarrow ((c1{,}s2))$
then have $c1{=}Skip$ **using** *stepc-elim-cases*(*4*) **by** *force*
} **thus** *?thesis* **by** *auto*
qed

lemma *no-comp-tran-before-i-spec*:
assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
a1: $fst\ (l!k) = Spec\ r\ e$ **and**
a2: $Suc\ i{<}length\ l \wedge k{\le}i \wedge (\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)))$ **and**
a3: $k{\le}j \wedge j < i \wedge (\Gamma\vdash_c(l!j)\ \rightarrow (l!(Suc\ j)))$ **and**
a4: $\forall k < j.\ (\Gamma\vdash_c(l!k)\ \rightarrow_e (l!(Suc\ k)))$ **and**
a5: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!0) \in Normal\ `\ p\ \wedge$
$Sta\ q\ rely \wedge snd\ (l!Suc\ j) \in Normal\ `\ q$
shows P

proof −
  have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1)\ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
    using *spec-skip* **by** *blast*
  thus *?thesis* **using** *a0 a1 a2 a3 a4 a5 no-comp-tran-before-i* **by** *blast*
qed

lemma *only-one-component-tran-i-spec*:
  assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
    a1: $fst\ (l!k) = Spec\ r\ e$ **and**
    a2: $Suc\ i<length\ l \wedge k\leq i \wedge (\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)))$ **and**
    a3: $k\leq j \wedge j \neq i \wedge Suc\ j < length\ l \wedge (\Gamma\vdash_c(l!j)\ \rightarrow (l!(Suc\ j))) \wedge fst\ (l!j) = Spec\ r\ e$ **and**
    a4: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!k) \in Normal\ `\ p \wedge Sta\ q\ rely \wedge snd\ (l!Suc\ j) \in Normal\ `\ q$
  shows P
proof −
  have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1)\ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
    using *spec-skip* **by** *blast*
  thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran-i*[*OF a0 a1 a2*] **by** *blast*
qed

lemma *only-one-component-tran-spec*:
  assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
    a1: $fst\ (l!k) = Spec\ r\ e$ **and**
    a2: $k\leq i\ \wedge i \neq j \wedge Suc\ i<length\ l \wedge (\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i))) \wedge fst\ (l!i) = Spec\ r\ e$ **and**
    a3: $k\leq j\ \wedge Suc\ j < length\ l$ **and**
    a4: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!k) \in Normal\ `\ p \wedge Sta\ q\ rely \wedge snd\ (l!Suc\ i) \in Normal\ `\ q$
  shows $(\Gamma\vdash_c(l!j)\ \rightarrow_e (l!(Suc\ j)))$
proof −
  have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1)\ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
    using *spec-skip* **by** *blast*
  thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran* **by** *blast*
qed

lemma *only-one-component-tran-all-env-spec*:
  assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
    a1: $fst\ (l!k) = Spec\ r\ e$ **and**
    a2: $k\leq i \wedge Suc\ i<length\ l \wedge(\Gamma\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)))\ \wedge fst\ (l!i) = Spec\ r\ e$ **and**
    a3: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!k) \in Normal\ `\ p \wedge Sta\ q\ rely \wedge snd\ (l!Suc\ i) \in Normal\ `\ q$
  shows $\forall j.\ k\leq j \wedge j\neq i \wedge Suc\ j < (length\ l) \longrightarrow (\Gamma\vdash_c(l!j)\ \rightarrow_e (l!(Suc\ j)))$
proof −
  have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1)\ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
    using *spec-skip* **by** *blast*
  thus *?thesis* **by** (*metis* (*no-types*) *a0 a1 a2 a3 only-one-component-tran-spec*)

qed

lemma *only-one-component-tran-all-not-comp-spec*:
assumes *a0*:$(\Gamma,\ l) \in$ *cptn* **and**
a1: *fst* $(l!k) =$ *Spec r e* **and**
a2: $k \leq i \land$ *Suc i<length l* $\land(\Gamma\vdash_c(l!i) \ \rightarrow (l!(Suc\ i))) \ \land$ *fst* $(l!i) =$ *Spec r e* **and**
a3: *env-tran-right* Γ *l rely* $\land$ *Sta p rely* $\land$ *snd* $(l!k) \in$ *Normal ' p* $\land$
Sta q rely $\land$ *snd* $(l!Suc\ i) \in$ *Normal ' q*
shows $\forall j.\ k \leq j \land j \neq i \land Suc\ j < (length\ l) \longrightarrow \neg(\Gamma\vdash_c(l!j) \ \rightarrow (l!(Suc\ j)))$
proof −
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1) \ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *spec-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 only-one-component-tran-all-not-comp* **by** *blast*
qed

lemma *one-component-tran-spec*:
assumes *a0*:$(\Gamma,\ l) \in$ *cptn* **and**
a1: *fst* $(l!0) =$ *Spec r e* **and**
a2: *Suc k<length l* $\land (\Gamma\vdash_c(l!k) \ \rightarrow (l!(Suc\ k)))$ **and**
a3: *env-tran-right* Γ *l rely* $\land$ *Sta p rely* $\land$ *snd* $(l!0) \in$ *Normal ' p* $\land$
Sta q rely **and**
a4:$p \subseteq \{s.\ (\forall t.\ (s,t)\in r \longrightarrow t \in q) \land (\exists t.\ (s,t) \in r)\}$

shows $\forall j.\ 0 \leq j \land j \neq k \land Suc\ j < (length\ l) \longrightarrow \neg(\Gamma\vdash_c(l!j) \ \rightarrow (l!(Suc\ j)))$
proof −
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1) \ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *spec-skip* **by** *blast*
also obtain *j* **where** *first*:(*Suc j<length l* $\land$ $(\Gamma\vdash_c(l!j) \ \rightarrow (l!Suc\ j))) \land$
$(\forall k < j.\ \neg((\Gamma\vdash_c(l!k) \ \rightarrow (l!(Suc\ k)))))$
by (*metis* (*no-types*) *a2 exist-first-comp-tran'*)
moreover then have *prg-j*:*fst* $(l!j) =$ *Spec r e* **using** *a1 a0*
by (*metis cptn-env-same-prog not-step-comp-step-env*)
moreover have *sta-j*:*snd* $(l!j) \in$ *Normal ' p*
proof −
have *a0'*:$0 \leq j \land j<(length\ l)$ **using** *first* **by** *auto*
have *a1'*:$(\forall k.\ 0 \leq k \land k < j \longrightarrow ((\Gamma\vdash_c(l!k) \ \rightarrow_e (l!(Suc\ k)))))$
using *first not-step-comp-step-env a0* **by** *fastforce*
thus *?thesis* **using** *stability first a3 a1' a0'* **by** *blast*
qed
then have *snd* $(l!Suc\ j) \in$ *Normal ' q* **using** *a4 first prg-j*
proof −
obtain *s* **where** *s*:*snd* $(l!j) =$ *Normal s* $\land$ $s \in p$ **using** *sta-j* **by** *fastforce*
then have *suc-skip*: *fst* $(l!Suc\ j) = Skip$
using *spec-skip first prg-j a4* **by** (*metis* (*no-types*, *lifting*) *prod.collapse*)
moreover obtain *s'* **where** *snd* $(l!Suc\ j) =$ *Normal s'* $\land (s,s')\in r$
proof −
{ **have** *f1*:$(\Gamma\vdash_c(fst(l!j),snd(l!j)) \ \rightarrow (fst(l!Suc\ j),snd(l!Suc\ j)))$ **using** *first* **by** *auto*

obtain t **where** *snd* $(l!Suc\ j) = Normal\ t$
using *step-spec-skip-normal-normal*[*of* Γ *fst*$(l!j)$ *snd*$(l!j)$ *fst*$(l!Suc\ j)$ *snd*$(l!Suc\ j)$ r]
suc-skip prg-j s a4 f1 **by** *blast*
moreover then have $(s,t)\in r$ **using** *a4 s prg-j f1 suc-skip stepc-Normal-elim-cases*(4)
by (*metis* (*no-types*, *lifting*) *stepc-Normal-elim-cases*(4) *prod.inject* *xstate.distinct*(5) *xstate.inject*(1))
ultimately have $\exists t.\ snd\ (l!Suc\ j)= Normal\ t \wedge (s,t)\in r$ **by** *auto*
}
then show $(\bigwedge s'.\ snd\ (l\ !\ Suc\ j) = Normal\ s' \wedge (s,\ s') \in r \Longrightarrow thesis) \Longrightarrow thesis$ **..**
qed
then show *?thesis* **using** *a4 sta-j s* **by** *auto*
qed
then have $\forall i.\ 0\leq i \wedge i\neq j \wedge Suc\ i < (length\ l) \longrightarrow \neg(\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$
using *only-one-component-tran-all-not-comp-spec*[*OF a0 a1*] *first a3*
a0 a1 calculation(1) *only-one-component-tran1 prg-j* **by** *blast*
moreover then have $k=j$ **using** *a2* **by** *fastforce*
ultimately show *?thesis* **by** *auto*
qed

lemma *one-component-tran-spec-env*:
assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
a1: *fst* $(l!0) = Spec\ r\ e$ **and**
a2: $Suc\ k<length\ l \wedge (\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))$ **and**
a3: *env-tran-right* Γ l *rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!0) \in Normal$ ‘ p $\wedge$
Sta q rely **and**
a4:$p \subseteq \{s.\ (\forall t.\ (s,t)\in r \longrightarrow t \in q) \wedge (\exists t.\ (s,t) \in r)\}$
shows $\forall j.\ 0\leq j \wedge j\neq k \wedge Suc\ j < (length\ l) \longrightarrow \Gamma\vdash_c(l!j) \rightarrow_e (l!(Suc\ j))$
proof –
have $\forall j.\ 0\leq j \wedge j\neq k \wedge Suc\ j < (length\ l) \longrightarrow \neg\ (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))$
using *one-component-tran-spec*[*OF a0 a1 a2 a3 a4*] **by** *auto*
thus *?thesis* **using** *a0*
by (*metis Suc-eq-plus1 cptn-tran-ce-i step-ce-elim-cases*)
qed

lemma *final-exist-component-tran-spec*:
assumes *a0*:$(\Gamma,\ l) \in cptn$ **and**
a1: *fst* $(l!i) = Spec\ r\ e$ **and**
a2: *env-tran* Γ q l R **and**
a3: $i\leq j \wedge j < length\ l \wedge final\ (l!j)$
shows $\exists k.\ k\geq i \wedge k<j \wedge (\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))$
proof –
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Spec\ r\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)$
using *spec-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 final-exist-component-tran* **by** *blast*
qed
lemma *Spec-sound1*:
$p \subseteq \{s.\ (\forall t.\ (s,t)\in r \longrightarrow t \in q) \wedge (\exists t.\ (s,t) \in r)\} \Longrightarrow$

$(\forall s\ t.\ s{\in}p\ \wedge (s,t){\in}r \longrightarrow (Normal\ s,\ Normal\ t) \in G) \Longrightarrow$
$Sta\ p\ R \Longrightarrow$
$Sta\ q\ R \Longrightarrow$
$c \in cp\ \Gamma\ (Spec\ r\ e)\ s \Longrightarrow$
$c \in assum(p,\ R) \Longrightarrow$
$c \in comm\ (G,\ (q,a))\ F$

proof −
assume
$a0{:}p \subseteq \{s.\ (\forall t.\ (s,t){\in}r \longrightarrow t \in q) \wedge (\exists t.\ (s,t) \in r)\}$ **and**
$a1{:}(\forall s\ t.\ s{\in}p\ \wedge (s,t){\in}r \longrightarrow (Normal\ s{,}Normal\ t) \in G)$ **and**
$a2{:}Sta\ p\ R$ **and**
$a3{:}Sta\ q\ R$ **and**
$a10{:}c \in cp\ \Gamma\ (Spec\ r\ e)\ s$ **and**
$a11{:}c \in assum(p,\ R)$

obtain $\Gamma 1\ l$ **where** $c\text{-}prod{:}c{=}(\Gamma 1{,}l)$ **by** *fastforce*
have $cp{:}l!0{=}(Spec\ r\ e{,}s) \wedge (\Gamma{,}l) \in cptn \wedge \Gamma{=}\Gamma 1$ **using** *a10 cp-def c-prod* **by** *fastforce*
have $assum{:}snd(l!0) \in Normal\ `\ (p) \wedge (\forall i.\ Suc\ i{<}length\ l \longrightarrow$
$(\Gamma 1){\vdash}_c(l!i)\ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
have $concl{:}(\forall i\ ns\ ns'.\ Suc\ i{<}length\ l \longrightarrow$
$\Gamma 1{\vdash}_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof −
{ fix k
assume $a00{:}Suc\ k{<}length\ l$ **and**
$a11{:}\Gamma 1{\vdash}_c(l!k)\ \rightarrow (l!(Suc\ k))$
obtain *ck sk csk ssk* **where** *tran-pair*:
$\Gamma 1{\vdash}_c\ (ck{,}sk)\ \rightarrow (csk,\ ssk) \wedge (ck = fst\ (l!k)) \wedge (sk = snd\ (l!k)) \wedge (csk = fst\ (l!(Suc\ k))) \wedge (ssk = snd\ (l!(Suc\ k)))$
using *a11* **by** *fastforce*
have $len\text{-}l{:}length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain $a\ l1$ **where** $l{:}l{=}a\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have $last\text{-}l{:}last\ l = l!(length\ l{-}1)$
using *last-length* [*of a l1*] *l* **by** *fastforce*
have *env-tran*:*env-tran* $\Gamma\ p\ l\ R$ **using** *assum env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* $\Gamma\ l\ R$
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
then have $all\text{-}event{:}\forall j.\ 0{\le}j \wedge j \neq k \wedge Suc\ j < length\ l \longrightarrow (\Gamma{\vdash}_c(l!j)\ \rightarrow_e (l!(Suc\ j)))$
using *a00 a11* *one-component-tran-spec-env*[*of* $\Gamma\ l\ r\ e\ k\ R$]
env-tran-right fst-conv a0 a2 a3 cp len-l assum
by *fastforce*
then have $before\text{-}k\text{-}all\text{-}evn{:}\forall j.\ 0{\le}j \wedge j < k\ \longrightarrow (\Gamma{\vdash}_c(l!j)\ \rightarrow_e (l!(Suc\ j)))$
using *a00 a11* **by** *fastforce*
then have $k\text{-}basic{:}ck = Spec\ r\ e \wedge sk \in Normal\ `\ (p)$
using *cp env-tran-right a2 assum a00 a11 stability*[*of p R l 0 k k* Γ] *tran-pair*

by *force*
have *suc-skip*: $csk = Skip$
using *a0 a00 k-basic tran-pair spec-skip* **by** *blast*
obtain s' **where** $ss{:}sk = Normal\ s' \wedge s' \in (p)$
using *k-basic* **by** *fastforce*
obtain s'' **where** $suc\text{-}k\text{-}skip\text{-}q{:}ssk = Normal\ s'' \wedge (s',s'') \in r$
proof −
{**from** *ss* **obtain** t **where** $ssk = Normal\ t$
using *step-spec-skip-normal-normal*[*of* $\Gamma 1$ *ck sk csk ssk r e s'*]
k-basic tran-pair a0 suc-skip
by *blast*
moreover then have $(s',t) \in r$ **using** *a0 k-basic ss a11 suc-skip*
by (*metis* (*no-types*, *lifting*) *stepc-Normal-elim-cases*(*4*) *tran-pair prod.inject xstate.distinct*(*5*) *xstate.inject*(*1*))
ultimately have $\exists t.\ ssk= Normal\ t\ \wedge (s',t) \in r$ **by** *auto*
}
then show $(\bigwedge s''.\ ssk = Normal\ s''\ \wedge (s',s'') \in r \Longrightarrow thesis) \Longrightarrow thesis$ **..**
qed
then have $(snd(l!k),\ snd(l!(Suc\ k))) \in G$
using *ss a1 tran-pair* **by** *force*
} **thus** *?thesis* **by** *auto* **qed**
have *concr*:$(final\ (last\ l) \longrightarrow ((fst\ (last\ l) = Skip\ \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw\ \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)))$
proof−
{
assume *valid*:*final* (*last l*)
have *len-l*:$length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a l1* **where** $l{:}l=a\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:$last\ l = l!(length\ l-1)$
using *last-length* [*of a l1*] *l* **by** *fastforce*
have *env-tran*:*env-tran* Γ *p l R* **using** *assum env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have $\exists k.\ k \geq 0 \wedge k<((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(Suc\ k)))$
proof −
have $0 \leq (length\ l-1)$ **using** *len-l last-l* **by** *auto*
moreover have $(length\ l-1) < length\ l$ **using** *len-l* **by** *auto*
moreover have $final\ (l!(length\ l-1))$ **using** *valid last-l* **by** *auto*
moreover have $fst\ (l!0) = Spec\ r\ e$ **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-spec env-tran* **by** *blast*
qed
then obtain k **where** *k-comp-tran*: $k \geq 0 \wedge k<((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(Suc\ k)))$
by *auto*
then obtain *ck sk csk ssk* **where** *tran-pair*:
$\Gamma 1 \vdash_c (ck,sk) \rightarrow (csk,\ ssk) \wedge (ck = fst\ (l!k)) \wedge (sk = snd\ (l!k)) \wedge (csk =$

fst (*l*!(*Suc k*))) ∧ (*ssk* = *snd* (*l*!(*Suc k*)))
using *cp* by *fastforce*
moreover then have *Suc k* < *length l* using *k-comp-tran* by *auto*
ultimately have *all-event*:∀ *j*. *0*≤*j* ∧ *j* ≠ *k* ∧ *Suc j* < *length l* ⟶ (Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*)))
using *one-component-tran-spec-env*[*of* Γ *l r e k R*] *a0 a11 a2 a3 assum cp env-tran-right fst-conv*
by *fastforce*
then have *before-k-all-evn*:∀ *j*. *0*≤*j* ∧ *j* < *k* ⟶ (Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*)))
using *k-comp-tran* by *fastforce*
then have *k-basic*:*ck* = *Spec r e* ∧ *sk* ∈ *Normal* ‘ (*p*)
using *cp env-tran-right a2 assum tran-pair k-comp-tran stability*[*of p R l 0 k k* Γ] *tran-pair*
by *force*
have *suc-skip*: *csk* = *Skip*
using *a0 k-basic tran-pair spec-skip* by *blast*
have *suc-k-skip-q*:*ssk* ∈ *Normal* ‘ *q*
proof −
obtain *s′* where *k-s*: *sk* =*Normal s′* ∧ *s′* ∈ (*p*)
using *k-basic* by *auto*
then obtain *t* where *ssk* = *Normal t*
using *step-spec-skip-normal-normal*[*of* Γ*1 ck sk csk ssk r*] *k-basic tran-pair a0 suc-skip*
by *blast*
then obtain *t* where *ssk*= *Normal t* by *fastforce*
then have (*s′*,*t*)∈*r* using *k-basic k-s a11 suc-skip*
by (*metis* (*no-types*, *lifting*) *stepc-Normal-elim-cases*(*4*) *tran-pair prod.inject xstate.distinct*(*5*) *xstate.inject*(*1*))
thus *ssk* ∈ *Normal* ‘ *q* using *a0 k-s* ‹*ssk* = *Normal t*› by *blast*
qed
have *after-k-all-evn*:∀ *j*. (*Suc k*)≤*j* ∧ *Suc j* < (*length l*) ⟶ (Γ⊢$_c$(*l*!*j*) →$_e$ (*l*!(*Suc j*)))
using *all-event k-comp-tran* by *fastforce*
then have *fst-last-skip*:*fst* (*last l*) = *Skip* ∧
snd ((*last l*)) ∈ *Normal* ‘ *q*
using *l tran-pair suc-skip last-l len-l cp*
env-tran-right a3 suc-k-skip-q
assum k-comp-tran stability [*of q R l Suc k* ((*length l*) − *1*) - Γ]
by (*metis One-nat-def Suc-eq-plus1 Suc-leI Suc-mono diff-Suc-1 lessI list.size*(*4*))

} thus *?thesis* by *auto* qed
note *res* = *conjI* [*OF concl concr*]
thus *?thesis* using *c-prod* unfolding *comm-def* by *auto*
qed

lemma *Spec-sound*:
p ⊆ {*s*. (∀ *t*. (*s*,*t*)∈*r* ⟶ *t* ∈ *q*) ∧ (∃ *t*. (*s*,*t*) ∈ *r*)} ⟹
(∀ *s t*. *s*∈*p* ∧ (*s*,*t*)∈*r* ⟶ (*Normal s*, *Normal t*) ∈ *G*) ⟹
Sta p R ⟹

$Sta\ q\ R \Longrightarrow$
$\Gamma,\Theta \models n_{/F}\ (Spec\ r\ e)\ sat\ [p,\ R,\ G,\ q,a]$
proof –
assume
$a0{:}p \subseteq \{s.\ (\forall t.\ (s,t){\in}r \longrightarrow t \in q) \wedge (\exists t.\ (s,t) \in r)\}$ **and**
$a1{:}(\forall s\ t.\ s{\in}p\ \wedge (s,t){\in}r \longrightarrow (Normal\ s,Normal\ t) \in G)$ **and**
$a2{:}Sta\ p\ R$ **and**
$a3{:}Sta\ q\ R$
{
fix s
have $cpn\ n\ \Gamma\ (Spec\ r\ e)\ s \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
proof –
{
fix c
assume $a10{:}c \in cpn\ n\ \Gamma\ (Spec\ r\ e)\ s$ **and** $a11{:}c \in assum(p,\ R)$
then have $a10{:}c{\in}cp\ \Gamma\ (Spec\ r\ e)\ s$
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
have $c{\in}comm(G,\ (q,a))\ F$ **using** *Spec-sound1*[*OF a0 a1 a2 a3 a10 a11*] **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

12.4 Await Sound

lemma *await-skip*:
$\forall s1\ s2\ c1.\ \Gamma\vdash_c(Await\ b\ c\ e,s1) \rightarrow ((c1,s2)) \longrightarrow c1{=}Skip \vee (c1 = Throw \wedge (\exists s21.\ s2 = Normal\ s21\ ))$
proof –
{**fix** $s1\ s2\ c1$
assume $\Gamma\vdash_c(Await\ b\ c\ e,s1) \rightarrow ((c1,s2))$
then have $c1{=}Skip \vee (c1 = Throw \wedge (\exists s21.\ s2 = Normal\ s21\ ))$ **using** *stepc-elim-cases*(*8*) **by** *blast*
} **thus** *?thesis* **by** *auto*
qed

lemma *no-comp-tran-before-i-await*:
assumes $a0{:}(\Gamma,\ l) \in cptn$ **and**
$a1{:}\ fst\ (l!k) = Await\ b\ c\ e$ **and**
$a2{:}\ Suc\ i{<}length\ l \wedge k{\le}i \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i)))$ **and**
$a3{:}\ k{\le}j \wedge j < i \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))$ **and**
$a4{:}\ \forall k < j.\ (\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k)))$ **and**
$a5{:}\ env\text{-}tran\text{-}right\ \Gamma\ l\ rely\ \wedge Sta\ p\ rely \wedge snd\ (l!0) \in Normal\ `\ p\ \wedge$
$Sta\ q\ rely \wedge snd\ (l!Suc\ j) \in Normal\ `\ q$
shows P
proof –
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Await\ b\ c\ e,s1) \rightarrow ((c1,s2)) \longrightarrow c1{=}Skip \vee (c1 = Throw$

$\wedge$ ($\exists$ *s21*. *s2* =*Normal s21*))
  using *await-skip* **by** *blast*
 thus *?thesis* **using** *a0 a1 a2 a3 a4 a5 no-comp-tran-before-i* **by** *blast*
qed

lemma *only-one-component-tran-i-await*:
 assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
  a1: *fst* (*l*!*k*) = *Await b c e* **and**
  a2: *Suc i*<*length l* $\wedge$ *k*$\leq$*i* $\wedge$ ($\Gamma\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*))) **and**
  a3: *k*$\leq$*j* $\wedge$ *j* $\neq$ *i* $\wedge$ *Suc j* < *length l* $\wedge$ ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow$ (*l*!(*Suc j*))) $\wedge$ *fst* (*l*!*j*) = *Await b c e* **and**
  a4: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*k*) $\in$ *Normal* ‘ *p* $\wedge$
   Sta q rely $\wedge$ *snd* (*l*!*Suc j*) $\in$ *Normal* ‘ *q*
 shows *P*
proof –
 have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c$(*Await b c e*,*s1*) $\rightarrow$ ((*c1*,*s2*)) $\longrightarrow$ (*c1*=*Skip*)$\vee$(*c1* = *Throw* $\wedge$ ($\exists$ *s21*. *s2* =*Normal s21*))
  using *await-skip* **by** *blast*
 thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran-i* **by** *blast*
qed

lemma *only-one-component-tran-await*:
 assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
  a1: *fst* (*l*!*k*) = *Await b c e* **and**
  a2: *k*$\leq$*i* $\wedge$ *i* $\neq$ *j* $\wedge$ *Suc i*<*length l* $\wedge$ ($\Gamma\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*))) $\wedge$ *fst* (*l*!*i*) = *Await b c e* **and**
  a3: *k*$\leq$*j* $\wedge$ *Suc j* < *length l* **and**
  a4: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*k*) $\in$ *Normal* ‘ *p* $\wedge$
   Sta q rely $\wedge$ *snd* (*l*!*Suc i*) $\in$ *Normal* ‘ *q*
 shows ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))
proof –
 have $\forall$ *s1 s2 c1*. $\Gamma\vdash_c$(*Await b c e*,*s1*) $\rightarrow$ ((*c1*,*s2*)) $\longrightarrow$ (*c1*=*Skip*)$\vee$(*c1* = *Throw* $\wedge$ ($\exists$ *s21*. *s2* =*Normal s21*))
  using *await-skip* **by** *blast*
 thus *?thesis* **using** *a0 a1 a2 a3 a4 only-one-component-tran* **by** *blast*
qed

lemma *only-one-component-tran-all-env-await*:
 assumes *a0*:(Γ, *l*) $\in$ *cptn* **and**
  a1: *fst* (*l*!*k*) = *Await b c e* **and**
  a2: *Suc i*<*length l* $\wedge$ *k*$\leq$*i* $\wedge$ ($\Gamma\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*))) $\wedge$ *fst* (*l*!*i*) = *Await b c e* **and**
  a3: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* (*l*!*k*) $\in$ *Normal* ‘ *p* $\wedge$
   Sta q rely $\wedge$ *snd* (*l*!*Suc i*) $\in$ *Normal* ‘ *q*
 shows $\forall$ *j*. *k*$\leq$*j* $\wedge$ *j*$\neq$*i* $\wedge$ *Suc j* < (*length l*) $\longrightarrow$ ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))
proof –
 have *a*:$\forall$ *s1 s2 c1*. $\Gamma\vdash_c$(*Await b c e*,*s1*) $\rightarrow$ ((*c1*,*s2*)) $\longrightarrow$ (*c1*=*Skip*)$\vee$(*c1* = *Throw*)
  using *await-skip* **by** *blast*

thus *?thesis* **by** (*metis* (*no-types*) *a0 a1 a2 a3 only-one-component-tran-await*)
qed

lemma *only-one-component-tran-all-not-comp-await*:
assumes *a0*:$(\Gamma, l) \in cptn$ **and**
a1: *fst* $(l!k) = Await\ b\ c\ e$ **and**
a2: $Suc\ i<length\ l \wedge k\leq i \wedge (\Gamma\vdash_c(l!i) \rightarrow (l!(Suc\ i))) \wedge fst\ (l!i) = Await\ b\ c\ e$ **and**
a3: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!k) \in Normal\ `\ p \wedge Sta\ q\ rely \wedge snd\ (l!Suc\ i) \in Normal\ `\ q$
shows $\forall j.\ k\leq j \wedge j\neq i \wedge Suc\ j < (length\ l) \longrightarrow \neg(\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))$
proof −
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Await\ b\ c\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)\vee(c1 = Throw \wedge (\exists s21.\ s2 =Normal\ s21\ ))$
using *await-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 only-one-component-tran-all-not-comp* **by** *blast*
qed

lemma *one-component-tran-await*:
assumes *a0*:$(\Gamma, l) \in cptn$ **and**
a1: *fst* $(l!0) = Await\ b\ c\ e$ **and**
a2: $Suc\ k<length\ l \wedge (\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))$ **and**
a3: $env\text{-}tran\text{-}right\ \Gamma\ l\ rely \wedge Sta\ p\ rely \wedge snd\ (l!0) \in Normal\ `\ p \wedge$
$Sta\ q\ rely \wedge$
$Sta\ a\ rely$ **and**
a4:$\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
$(p \cap b \cap \{V\})\ c$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q),$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a)$ **and**
a5:$snd\ (last\ l) \notin Fault\ `\ F$

shows $(\forall j.\ 0\leq j \wedge j\neq k \wedge Suc\ j < (length\ l) \longrightarrow \neg(\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))) \wedge$
$(\exists s\ s'.\ fst\ (l!k) = Await\ b\ c\ e \wedge snd\ (l!k) \in Normal\ `\ (p) \wedge snd\ (l!k) = Normal\ s \wedge snd\ (l!Suc\ k) = Normal\ s' \wedge$
$(snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q) \vee$
$snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)))$
proof −
have *suc-skip*:$\forall s1\ s2\ c1.\ \Gamma\vdash_c(Await\ b\ c\ e,s1) \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)\vee (c1=Throw \wedge (\exists s21.\ s2 =Normal\ s21\ ))$
using *await-skip* **by** *blast*
also obtain j **where** *first*:$(Suc\ j<length\ l \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j)))) \wedge$
$(\forall k < j.\ \neg((\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))))$
by (*metis* (*no-types*) *a2 exist-first-comp-tran'*)
moreover then have *prg-j*:$fst\ (l!j) = Await\ b\ c\ e$ **using** *a1 a0*
by (*metis cptn-env-same-prog not-step-comp-step-env*)
moreover have *sta-j*:$snd\ (l!j) \in Normal\ `\ p$
proof −
have *a0'*:$0\leq j \wedge j<(length\ l)$ **using** *first* **by** *auto*

have $a1'$:($\forall k.\ 0 \leq k \wedge k < j \longrightarrow ((\Gamma\vdash_c(l!k) \ \rightarrow_e (l!(Suc\ k)))))$
using *first not-step-comp-step-env a0* **by** *fastforce*
thus *?thesis* **using** *stability first a3 a1' a0'* **by** *blast*
qed
from *sta-j* **obtain** *s* **where**
k-basic:$fst\ (l!j) = Await\ b\ c\ e \wedge snd\ (l!j) = Normal\ s \wedge\ s \in p \wedge snd(l!j) \in Normal\ `\ p$
using *sta-j prg-j* **by** *fastforce*
then have *conc*:$snd\ (l!Suc\ j) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q) \vee$
$snd\ (l!Suc\ j) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)$
proof –
have $\Gamma_{\neg a},\{\}\models_{/F}$
$(p \cap b \cap \{s\})\ c$
$(\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q),$
$(\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)$
using *a4 hoare-sound* **by** *fastforce*
then have *e-auto*:$\Gamma_{\neg a}\models_{/F}\ (p \cap b \cap \{s\})\ c$
$(\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q),$
$(\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)$
unfolding *cvalid-def* **by** *auto*
have f': $\Gamma\vdash_c(fst\ (l!j),\ snd(l!j)) \ \rightarrow (fst(l!(Suc\ j)),\ snd(l!(Suc\ j)))$
using *first* **by** *auto*
have *step-await*:$Suc\ j<length\ l \wedge \Gamma\vdash_c\ (Await\ b\ c\ e,snd(l!j)) \ \rightarrow (fst(l!(Suc\ j)),\ snd(l!(Suc\ j)))$
using *f' k-basic first* **by** *fastforce*
then have *s'-in-bp*:$s\in\ b \wedge\ s \in\ p$ **using** *k-basic stepc-Normal-elim-cases(8)* **by** *metis*
then have $s \in (p \cap b)$ **by** *fastforce*
moreover have *test*:
$\exists t.\ \Gamma_{\neg a}\vdash \langle c,Normal\ s\rangle \Rightarrow t\ \wedge$
$((\exists t'.\ t\ =Abrupt\ t' \wedge snd(l!Suc\ j) = Normal\ t') \vee$
$(\forall t'.\ t \neq Abrupt\ t' \wedge snd(l!Suc\ j)=t))$
proof –
fix *t*
{ assume $fst(l!Suc\ j) = Skip$
then have *step*:$\Gamma\vdash_c\ (Await\ b\ c\ e,Normal\ s) \ \rightarrow (Skip,\ snd(l!Suc\ j))$
using *step-await k-basic* **by** *fastforce*
have *s'-b*:$s\in b$ **using** *s'-in-bp* **by** *fastforce*
note *step = stepc-elim-cases-Await-skip[OF step]*
have *h*:$(s \in\ b \Longrightarrow \Gamma_{\neg a}\vdash \langle c,Normal\ s\rangle \Rightarrow snd(l!Suc\ j) \Longrightarrow \forall t'.\ snd(l!Suc\ j) \neq Abrupt\ t' \Longrightarrow$
$\Gamma_{\neg a}\vdash \langle c,Normal\ s\rangle \Rightarrow snd(l!Suc\ j) \wedge (\forall t'.\ snd(l!Suc\ j) \neq Abrupt\ t'))$ **by** *auto*
have *?thesis*
using *step[OF h]* **by** *fastforce*
} **note** *left = this*
{ assume $fst(l!Suc\ j)=\ Throw \wedge (\exists s1.\ snd(l!Suc\ j) = Normal\ s1)$
then obtain *s1* **where** *step*:$fst(l!Suc\ j)=\ Throw \wedge snd(l!Suc\ j) = Normal$

s1
by *fastforce*
then have *step*: $\Gamma\vdash_c$ (*Await b c e*,*Normal s*) $\rightarrow$ (*Throw*, *snd*(*l*!*Suc j*))
using *step-await k-basic* by *fastforce*
have *s'-b*:*s*∈*b* using *s'-in-bp* by *fastforce*
note *step* = *stepc-elim-cases-Await-throw*[*OF step*]
have *h*:($\bigwedge t'$. *snd*(*l*!*Suc j*) = *Normal t'* $\Longrightarrow$ *s* ∈ *b* $\Longrightarrow$ $\Gamma_{\neg a}\vdash$ ⟨*c*,*Normal s*⟩ ⇒ *Abrupt t'* $\Longrightarrow$
$\Gamma_{\neg a}\vdash$ ⟨*c*,*Normal s*⟩ ⇒ *Abrupt t'* ∧ *snd*(*l*!*Suc j*) = *Normal t'*)
by *auto*
have *?thesis* using *step*[*OF h*] by *blast*
} thus *?thesis* using *suc-skip left step-await suc-skip* by *blast*
qed
then obtain *t* where *e-step*:$\Gamma_{\neg a}\vdash$ ⟨*c*,*Normal s*⟩ ⇒ *t* ∧
((∃ *t'*. *t* =*Abrupt t'* ∧ *snd*(*l*!*Suc j*) = *Normal t'*) ∨
(∀ *t'*. *t* ≠ *Abrupt t'* ∧ *snd*(*l*!*Suc j*)=*t*)) by *fastforce*
moreover have *t* ∉ *Fault* ‘ *F*
proof −
{assume *a10*:*t* ∈ *Fault* ‘ *F*
then obtain *tf* where *t*=*Fault tf* ∧ *tf*∈*F* by *fastforce*
then have *snd*(*l*!*Suc j*) = *Fault tf* ∧ *tf*∈*F* using *e-step* by *fastforce*
also have *snd*(*l*!*Suc j*) ∉ *Fault* ‘ *F*
using *last-not-F*[*of* Γ *l F*] *a5 a1 step-await a0* by *blast*
ultimately have *False* by *auto*
} thus *?thesis* by *auto*
qed
ultimately have *t-q-a*:*t* ∈ *Normal* ‘ ({*s'*. (*Normal s*, *Normal s'*) ∈ *G*} ∩*q*) ∪
Abrupt ‘ ({*s'*. (*Normal s*, *Normal s'*) ∈ *G*} ∩ *a*)
using *e-auto* unfolding *valid-def* by *fastforce*
thus *?thesis* using *e-step t-q-a* by *blast*
qed
then have ∀ *i*. *0*≤*i* ∧ *i*≠*j* ∧ *Suc i* < (*length l*) ⟶ ¬($\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*)))
using *only-one-component-tran-all-not-comp-await*[*OF a0 a1*] *first a3*
a0 a1 calculation(*1*) *only-one-component-tran1 prg-j* by *blast*
moreover then have *k*:*k*=*j* using *a2* by *fastforce*
ultimately have (∀ *j*. *0*≤*j* ∧ *j*≠*k* ∧ *Suc j* < (*length l*) ⟶ ¬($\Gamma\vdash_c$(*l*!*j*) → (*l*!(*Suc j*)))) by *auto*
also from *conc k k-basic* have
(∃ *s s'*. *fst* (*l*!*k*) = *Await b c e* ∧ *snd* (*l*!*k*) ∈ *Normal* ‘ (*p*) ∧ *snd* (*l*!*k*) = *Normal s* ∧ *snd* (*l*!*Suc k*) = *Normal s'* ∧
(*snd* (*l*!*Suc k*) ∈ *Normal* ‘ ({*s'*. (*Normal s*, *Normal s'*) ∈ *G*} ∩ *q*) ∨
snd (*l*!*Suc k*) ∈ *Normal* ‘ ({*s'*. (*Normal s*, *Normal s'*) ∈ *G*} ∩ *a*)))
by *fastforce*
ultimately show *?thesis* by *auto*
qed

lemma *one-component-tran-await-env*:
assumes *a0*:(Γ, *l*) ∈ *cptn* and
a1: *fst* (*l*!*0*) = *Await b c e* and

$a2$: *Suc* k<*length* $l \wedge (\Gamma\vdash_c(l!k) \ \rightarrow (l!(Suc\ k)))$ **and**
$a3$: *env-tran-right* Γ *l rely* $\wedge$ *Sta p rely* $\wedge$ *snd* $(l!0) \in$ *Normal* ' $p \wedge$
Sta q rely $\wedge$
Sta a rely **and**
$a4$:$\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
$(p \cap b \cap \{V\})\ c$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q)$,
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a)$ **and**
$a5$:*snd* (*last* l) $\notin$ *Fault* ' F
shows $(\forall j.\ 0\leq j \wedge j\neq k \wedge Suc\ j < (length\ l) \longrightarrow (\Gamma\vdash_c(l!j) \ \rightarrow_e (l!(Suc\ j)))) \wedge$
$(\exists s\ s'.\ fst\ (l!k) = Await\ b\ c\ e \wedge snd\ (l!k) \in Normal\ `\ (p) \wedge$
$snd\ (l!k) = Normal\ s \wedge snd\ (l!Suc\ k) = Normal\ s' \wedge$
$(snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q) \vee$
$snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)))$
proof –
have $(\forall j.\ 0\leq j \wedge j\neq k \wedge Suc\ j < (length\ l) \longrightarrow \neg\ (\Gamma\vdash_c(l!j) \ \rightarrow (l!(Suc\ j)))) \wedge$
$(\exists s\ s'.\ fst\ (l!k) = Await\ b\ c\ e \wedge snd\ (l!k) \in Normal\ `\ (p) \wedge$
$snd\ (l!k) = Normal\ s \wedge snd\ (l!Suc\ k) = Normal\ s' \wedge$
$(snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q) \vee$
$snd\ (l!Suc\ k) \in Normal\ `\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)))$
using *one-component-tran-await*[*OF a0 a1 a2 a3 a4 a5*] **by** *auto*
thus *?thesis* **using** *a0*
by (*metis Suc-eq-plus1 cptn-tran-ce-i step-ce-elim-cases*)
qed

lemma *final-exist-component-tran-await*:
assumes $a0$:$(\Gamma,\ l) \in cptn$ **and**
$a1$: $fst\ (l!i) = Await\ b\ c\ e$ **and**
$a2$: *env-tran* Γ *q l R* **and**
$a3$: $i\leq j \wedge j < length\ l \wedge final\ (l!j)$
shows $\exists k.\ k\geq i \wedge k<j \wedge (\Gamma\vdash_c(l!k) \ \rightarrow (l!(Suc\ k)))$
proof –
have $\forall s1\ s2\ c1.\ \Gamma\vdash_c(Await\ b\ c\ e,s1) \ \rightarrow ((c1,s2)) \longrightarrow (c1=Skip)\vee(c1 = Throw$
$\wedge (\exists s21.\ s2 =Normal\ s21\ ))$
using *await-skip* **by** *blast*
thus *?thesis* **using** *a0 a1 a2 a3 final-exist-component-tran* **by** *blast*
qed

inductive-cases *stepc-elim-cases-Await-Fault*:
$\Gamma\vdash_c\ (Await\ b\ c\ e,\ Normal\ s) \rightarrow (u,Fault\ f)$

lemma *Await-sound1*:
$\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
$(p \cap b \cap \{V\})\ e$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q)$,
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a) \Longrightarrow$
Sta p R $\Longrightarrow$ *Sta q R* $\Longrightarrow$ *Sta a R* $\Longrightarrow$
$c \in cp\ \Gamma\ (Await\ b\ e\ e1)\ s \Longrightarrow$
$c \in assum(p,\ R) \Longrightarrow$

$c \in comm\ (G,\ (q,a))\ F$
proof −
assume
$a0$: $\forall\, V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
$(p \cap b \cap \{V\})\ e$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q),$
$(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a)$ **and**
$a2$:$Sta\ p\ R$ **and**
$a3$:$Sta\ q\ R$ **and**
$a4$:$Sta\ a\ R$ **and**
$a10$:$c \in cp\ \Gamma\ (Await\ b\ e\ e1)\ s$ **and**
$a11$:$c \in assum(p,\ R)$

obtain $\Gamma 1\ l$ **where** $c\text{-}prod$:$c=(\Gamma 1,l)$ **by** *fastforce*
{**assume** $last\text{-}fault$:$snd\ (last\ l) \notin Fault\ \text{‘}\ F$
have cp:$l!0=(Await\ b\ e\ e1,s) \wedge (\Gamma,l) \in cptn \wedge \Gamma=\Gamma 1$ **using** *a10 cp-def c-prod* **by** *fastforce*
have $assum$:$snd(l!0) \in Normal\ \text{‘}\ (p) \wedge (\forall\, i.\ Suc\ i<length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
have $concl$:$(\forall\, i\ ns\ ns'.\ Suc\ i<length\ l \longrightarrow$
$\Gamma 1\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof −
{ **fix** $k\ ns\ ns'$
assume $a00$:$Suc\ k<length\ l$ **and**
$a11$:$\Gamma 1\vdash_c(l!k)\ \rightarrow (l!(Suc\ k))$
have $len\text{-}l$:$length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain $a1\ l1$ **where** l:$l=a1\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have $env\text{-}tran$:$env\text{-}tran\ \Gamma\ p\ l\ R$ **using** *assum env-tran-def cp* **by** *blast*
then have $env\text{-}tran\text{-}right$: $env\text{-}tran\text{-}right\ \Gamma\ l\ R$
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
then have $all\text{-}event$:
$(\exists\, s\ s'.\ fst\ (l!k) = Await\ b\ e\ e1 \wedge snd\ (l!k) \in Normal\ \text{‘}\ (p) \wedge snd\ (l!k) =$
$Normal\ s \wedge snd\ (l!Suc\ k) = Normal\ s' \wedge$
$(snd\ (l!Suc\ k) \in Normal\ \text{‘}\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap q) \vee$
$snd\ (l!Suc\ k) \in Normal\ \text{‘}\ (\{s'.\ (Normal\ s,\ Normal\ s') \in G\} \cap a)))$
using *a00 a11* $one\text{-}component\text{-}tran\text{-}await\text{-}env[of\ \Gamma\ l\ b\ e\ e1\ k\ R\ p\ q\ a\ F\ G]$ *env-tran-right cp len-l*
using *a0 a2 a3 a4 assum fst-conv last-fault* **by** *auto*
then obtain $s'\ s''$ **where** ss:
$snd\ (l!k) = Normal\ s' \wedge s' \in (p) \wedge\ snd\ (l!Suc\ k) = Normal\ s''$
$\wedge (s'' \in ((\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap q)) \vee$
$s''\in ((\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap a)))$
by *fastforce*
then have $(snd(l!k),\ snd(l!(Suc\ k))) \in G$
using *a2* **by** *force*
} **thus** *?thesis* **using** *c-prod* **by** *auto* **qed**

have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ (*a*)))
proof−
{
assume *valid*:*final* (*last l*)
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:*l*=*a1*#*l1* **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
have *env-tran*:*env-tran* Γ *p l R* **using** *assum env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have ∃*k*. *k*≥*0* ∧ *k*<((*length l*) − *1*) ∧ ($\Gamma\vdash_c$(*l*!*k*) → (*l*!(*Suc k*)))
proof −
have *0*≤ (*length l*−*1*) **using** *len-l last-l* **by** *auto*
moreover have (*length l*−*1*) < *length l* **using** *len-l* **by** *auto*
moreover have *final* (*l*!(*length l*−*1*)) **using** *valid last-l* **by** *auto*
moreover have *fst* (*l*!*0*) = *Await b e e1* **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-await env-tran* **by** *blast*
qed
then obtain *k* **where** *k-comp-tran*: *k*≥*0* ∧ *Suc k* < *length l* ∧ ($\Gamma\vdash_c$(*l*!*k*) → (*l*!(*Suc k*)))
by *fastforce*
then obtain *ck sk csk ssk* **where** *tran-pair*:
$\Gamma 1\vdash_c$ (*ck*,*sk*) → (*csk*, *ssk*) ∧ (*ck* = *fst* (*l*!*k*)) ∧ (*sk* = *snd* (*l*!*k*)) ∧ (*csk* = *fst* (*l*!(*Suc k*))) ∧ (*ssk* = *snd* (*l*!(*Suc k*)))
using *cp* **by** *fastforce*
have *all-event*:
(∀*j*. *0*≤*j* ∧ *j*≠*k* ∧ *Suc j* < (*length l*) ⟶ ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))) ∧
(∃*s s*′. *fst* (*l*!*k*) = *Await b e e1* ∧ *snd* (*l*!*k*) ∈ *Normal* ‘ (*p*) ∧ *snd* (*l*!*k*) =
Normal s ∧ *snd* (*l*!*Suc k*) = *Normal s*′ ∧
(*snd* (*l*!*Suc k*) ∈ *Normal* ‘ ({*s*′. (*Normal s*, *Normal s*′) ∈ *G*} ∩ *q*) ∨
snd (*l*!*Suc k*) ∈ *Normal* ‘ ({*s*′. (*Normal s*, *Normal s*′) ∈ *G*} ∩ *a*)))
using *one-component-tran-await-env*[*of* Γ *l b e e1 k R p q a F G*] *a0 a11 a2 a3 a4 assum cp*
env-tran-right len-l fst-conv last-fault k-comp-tran **by** *fastforce*
then have *before-k-all-evn*:∀*j*. *0*≤*j* ∧ *j* < *k* ⟶ ($\Gamma\vdash_c$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*)))
using *k-comp-tran* **by** *fastforce*
then obtain *s*′ **where** *k-basic*:*ck* = *Await b e e1* ∧ *sk* ∈ *Normal* ‘ (*p*) ∧ *sk* = *Normal s*′
using *cp env-tran-right a2 assum tran-pair k-comp-tran stability*[*of p R l 0 k k* Γ] *tran-pair*
by *force*
have $\Gamma_{\neg a}$,{}$\models_{/F}$
(*p* ∩ *b* ∩ {*s*′}) *e*

$(\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap q)$,
$(\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap a)$
using *a0 hoare-sound k-basic*
by *fastforce*
then have *e-auto*:$\Gamma_{\neg a}\models_{/F}$ $(p \cap b \cap \{s'\})$ *e*
$(\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap q)$,
$(\{s.\ (Normal\ s',\ Normal\ s) \in G\} \cap a)$
unfolding *cvalid-def* **by** *auto*
have *after-k-all-evn*:$\forall j.\ (Suc\ k) \leq j \wedge Suc\ j < (length\ l) \longrightarrow (\Gamma\vdash_c(l!j) \rightarrow_e (l!(Suc\ j)))$
using *all-event k-comp-tran* **by** *fastforce*
have *suc-skip*: $csk = Skip \vee (csk = Throw \wedge (\exists s1.\ ssk = Normal\ s1))$
using *a0 k-basic tran-pair await-skip* **by** *blast*
moreover {
assume *at*:$csk = Skip$
then have *atom-tran*:$\Gamma_{\neg a}\vdash\langle e, sk\rangle \Rightarrow ssk$
using *k-basic tran-pair k-basic cp stepc-elim-cases-Await-skip*
by *metis*
have *sk-in-normal-pb*:$sk \in Normal$ ‘ $(p \cap b)$
using *k-basic tran-pair at cp stepc-elim-cases-Await-skip*
by (*metis* (*no-types*, *lifting*) *IntI image-iff*)
then have *fst* (*last l*) = *Skip* $\wedge$
snd ((*last l*)) $\in$ *Normal* ‘ *q*
proof (*cases ssk*)
case (*Normal t*)
then have $ssk \in Normal$ ‘ *q*
using *sk-in-normal-pb k-basic e-auto Normal atom-tran* **unfolding** *valid-def*
by *blast*
thus *?thesis*
using *at l tran-pair last-l len-l cp*
env-tran-right a3 after-k-all-evn
assum k-comp-tran stability [*of q R l Suc k* ((*length l*) − *1*) - Γ]
by (*metis* (*no-types*, *hide-lams*) *Suc-leI diff-Suc-eq-diff-pred diff-less less-one zero-less-diff*)
next
case (*Abrupt t*)
thus *?thesis*
using *at k-basic tran-pair k-basic cp stepc-elim-cases-Await-skip*
by *metis*
next
case (*Fault f1*)
then have $ssk \in Normal$ ‘ $q \vee ssk \in Fault$ ‘ *F*
using *k-basic sk-in-normal-pb e-auto Fault atom-tran* **unfolding** *valid-def* **by** *auto*
thus *?thesis*
proof
assume $ssk \in Normal$ ‘ *q* **thus** *?thesis* **using** *Fault* **by** *auto*
next
assume *suck-fault*:$ssk \in Fault$ ‘ *F*

have $\forall\, i<length\ l.\ snd\ (l\ !\ i) \notin Fault$ ' F
using *last-not-F*[*of* Γ *l F*] *last-fault cp* **by** *auto*
thus *?thesis*
using *cp tran-pair a11 k-comp-tran suck-fault*
by (*meson diff-less len-l less-imp-Suc-add less-one less-trans-Suc*)

qed
next
case (*Stuck*)
then have *ssk* ∈ *Normal* ' *q*
using *k-basic sk-in-normal-pb e-auto Stuck atom-tran* **unfolding** *valid-def*
by *blast*
thus *?thesis* **using** *Stuck* **by** *auto*
qed
}
moreover {
assume *at*:(*csk* = *Throw* ∧ (∃ *t*. *ssk* = *Normal t*))
then obtain *t* **where** *ssk-normal*:*ssk*=*Normal t* **by** *auto*
then have *atom-tran*:$\Gamma_{\neg a}\vdash\langle e,sk\rangle \Rightarrow$ *Abrupt t*
using *at k-basic tran-pair k-basic ssk-normal cp stepc-elim-cases-Await-throw xstate.inject*(*1*)
by *metis*
also have *sk* ∈ *Normal* ' (*p* ∩ *b*)
using *k-basic tran-pair k-basic ssk-normal at cp stepc-elim-cases-Await-throw*
by (*metis* (*no-types*, *lifting*) *IntI imageE image-eqI stepc-elim-cases-Await-throw*)

then have *ssk* ∈ *Normal* ' *a*
using *e-auto k-basic ssk-normal atom-tran* **unfolding** *valid-def*
by *blast*
then have (*fst* (*last l*) = *Throw* ∧ *snd* (*last l*) ∈ *Normal* ' (*a*))
using *at l tran-pair last-l len-l cp*
env-tran-right a4 after-k-all-evn
assum k-comp-tran stability [*of a R l Suc k* ((*length l*) − *1*) - Γ]
by (*metis* (*no-types*, *hide-lams*) *Suc-leI diff-Suc-eq-diff-pred diff-less less-one zero-less-diff*)
}
ultimately have *fst* (*last l*) = *Skip* ∧
snd ((*last l*)) ∈ *Normal* ' *q* ∨
(*fst* (*last l*) = *Throw* ∧ *snd* (*last l*) ∈ *Normal* ' (*a*))
by *blast*
} thus *?thesis* **by** *auto* **qed**
note *res* = *conjI* [*OF concl concr*]
}
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *auto*
qed

12.5 If sound

lemma *cptn-assum-induct*:

assumes
  a0: $(\Gamma,l) \in (cp\ \Gamma\ c\ s) \wedge ((\Gamma,l) \in assum(p,\ R))$ **and**
  a1: $k < length\ l \wedge l!k=(c1,Normal\ s') \wedge s' \in p1$
shows $(\Gamma,drop\ k\ l)\in ((cp\ \Gamma\ c1\ (Normal\ s')) \cap assum(p1,\ R)\ )$
proof −
  have *drop-k-s*:$(drop\ k\ l)!0 = (c1,Normal\ s')$ **using** *a1* **by** *fastforce*
  have *p1*:$s' \in p1$ **using** *a1* **by** *auto*
  have *k-l*:$k < length\ l$ **using** *a1* **by** *auto*
  show *?thesis*
  proof
    show $(\Gamma,\ drop\ k\ l) \in cp\ \Gamma\ c1\ (Normal\ s')$
    unfolding *cp-def*
    using *dropcptn-is-cptn a0 a1 drop-k-s cp-def*
    by *fastforce*
  next
    let *?c*= $(\Gamma,drop\ k\ l)$
    have *l*:$snd((snd\ ?c!0)) \in Normal\ `\ p1$
     using *p1 drop-k-s* **by** *auto*
    {**fix** *i*
     assume *a00*:$Suc\ i<length\ (snd\ ?c)$
     assume *a11*:$(fst\ ?c)\vdash_c((snd\ ?c)!i)\ \rightarrow_e\ ((snd\ ?c)!(Suc\ i))$
     have $(snd((snd\ ?c)!i),\ snd((snd\ ?c)!(Suc\ i))) \in R$
     using *a0* **unfolding** *assum-def* **using** *a00 a11* **by** *auto*
    } **thus** $(\Gamma,\ drop\ k\ l) \in assum\ (p1,\ R)$
      using *l* **unfolding** *assum-def* **by** *fastforce*
  qed
qed

lemma *Await-sound*:
 $\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
    $(p \cap b \cap \{V\})\ e$
    $(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q)$,
    $(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a) \Longrightarrow$
 $Sta\ p\ R \Longrightarrow Sta\ q\ R \Longrightarrow Sta\ a\ R \Longrightarrow$
 $\Gamma,\Theta \models n_{/F}\ \ (Await\ b\ e\ e1)\ sat\ [p,\ R,\ G,\ q,a]$
proof −
 assume
  a0: $\forall V.\ \Gamma_{\neg a},\{\}\vdash_{/F}$
       $(p \cap b \cap \{V\})\ e$
       $(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap q)$,
       $(\{s.\ (Normal\ V,\ Normal\ s) \in G\} \cap a)$ **and**
  a2:$Sta\ p\ R$ **and**
  a3:$Sta\ q\ R$ **and**
  a4:$Sta\ a\ R$
{
    fix *s*
    have $cpn\ n\ \Gamma\ (Await\ b\ e\ e1)\ s \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
    proof −
    {

fix *c*
assume *a10*:$c \in$ *cpn n* Γ (*Await b e e1*) *s* **and** *a11*:$c \in$ *assum*(*p*, *R*)
then have *a10*:$c{\in}$*cp* Γ (*Await b e e1*) *s*
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
have $c{\in}$*comm*(*G*, (*q*,*a*)) *F* **using** *Await-sound1*[*OF a0 a2 a3 a4 a10 a11*] **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *cptn-comm-induct*:
assumes
a0: (Γ,*l*) $\in$ (*cp* Γ *c s*) **and**
a1: *l1* = *drop j l* $\wedge$ (Γ, *l1*)$\in$ *comm*(*G*, (*q*,*a*)) *F* **and**
a2: $k \geq j \wedge j <$ *length l*
shows *snd* (*last* (*l*)) $\notin$ *Fault* ‘ *F* $\longrightarrow$ ((*Suc k* < *length l* $\longrightarrow$
$\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*)) $\longrightarrow$
(*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) $\in$ *G*)
$\wedge$ (*final* (*last* (*l*)) $\longrightarrow$
((*fst* (*last* (*l*)) = *Skip* $\wedge$
snd (*last* (*l*)) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last* (*l*)) = *Throw* $\wedge$
snd (*last* (*l*)) $\in$ *Normal* ‘ (*a*))))
proof −
have *pair-*Γ*l*:*fst* (Γ,*l1*) = Γ $\wedge$ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
have *a03*:*snd* (*last* (*l1*)) $\notin$ *Fault* ‘ *F* $\longrightarrow$($\forall$ *i*.
Suc i<*length* (*snd* (Γ, *l1*)) $\longrightarrow$
fst (Γ, *l1*)$\vdash_c$((*snd* (Γ, *l1*))!*i*) $\rightarrow$ ((*snd* (Γ, *l1*))!(*Suc i*)) $\longrightarrow$

(*snd*((*snd* (Γ, *l1*))!*i*), *snd*((*snd* (Γ, *l1*))!(*Suc i*))) $\in$ *G*) $\wedge$
(*final* (*last* (*snd* (Γ, *l1*))) $\longrightarrow$
snd (*last* (*snd* (Γ, *l1*))) $\notin$ *Fault* ‘ *F* $\longrightarrow$
((*fst* (*last* (*snd* (Γ, *l1*))) = *Skip* $\wedge$
snd (*last* (*snd* (Γ, *l1*))) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last* (*snd* (Γ, *l1*))) = *Throw* $\wedge$
snd (*last* (*snd* (Γ, *l1*))) $\in$ *Normal* ‘ (*a*)))
using *a1* **unfolding** *comm-def* **by** *fastforce*
have *last-l*:*last l1* = *last l* **using** *a1 a2* **by** *fastforce*
show *?thesis*
proof −
{
assume *snd* (*last l*) $\notin$ *Fault* ‘ *F*
then have *l1-f*:*snd* (*last l1*) $\notin$ *Fault* ‘ *F*
using *a03 a1 a2* **by** *force*
{ **assume** *Suc k* < *length l*
then have *a2*: $k \geq j \wedge$ *Suc k* < *length l* **using** *a2* **by** *auto*

have $k \leq$ *length l* **using** *a2* **by** *fastforce*
then have *l1-l*:$(l!k = l1!\ (k - j)\ ) \wedge (l!Suc\ k = l1!Suc\ (k - j))$
using *a1 a2* **by** *fastforce*
have *a00*:$Suc\ (k - j) <$ *length l1* **using** *a1 a2* **by** *fastforce*
have $\Gamma\vdash_c(l1!(k-j)) \rightarrow (l1!(Suc\ (k-j))) \longrightarrow$
$(snd((snd\ (\Gamma,\ l1))!(k-j)),\ snd((snd\ (\Gamma,\ l1))!(Suc\ (k-j)))) \in G$
using *pair-Γl a00 l1-f a03* **by** *presburger*
then have $\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)) \longrightarrow$
$(snd\ (l\ !\ k),\ snd\ (l\ !\ Suc\ k)) \in G$
using *l1-l last-l* **by** *auto*
} **then have** *l-side*:$Suc\ k <$ *length l* $\longrightarrow$
$\Gamma\vdash_c\ l\ !\ k \rightarrow l\ !\ Suc\ k \longrightarrow$
$(snd\ (l\ !\ k),\ snd\ (l\ !\ Suc\ k)) \in G$ **by** *auto*
{
assume *a10*:*final* (*last* (*l*))
then have *final-eq*: *final* (*last* (*l1*))
using *a10 a1 a2* **by** *fastforce*
also have *snd* (*last* (*l1*)) $\notin$ *Fault* ‘ *F*
using *last-l l1-f* **by** *fastforce*
ultimately have ((*fst* (*last* (*snd* (Γ, *l1*))) = *Skip* ∧
snd (*last* (*snd* (Γ, *l1*))) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last* (*snd* (Γ, *l1*))) = *Throw* ∧
snd (*last* (*snd* (Γ, *l1*))) ∈ *Normal* ‘ (*a*))
using *pair-Γl a03* **by** *presburger*
then have ((*fst* (*last* (*snd* (Γ, *l*))) = *Skip* ∧
snd (*last* (*snd* (Γ, *l*))) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last* (*snd* (Γ, *l*))) = *Throw* ∧
snd (*last* (*snd* (Γ, *l*))) ∈ *Normal* ‘ (*a*))
using *final-eq a1 a2* **by** *auto*
} **then have**
r-side:
SmallStepCon.final (*last l*) ⟶
fst (*last l*) = *LanguageCon.com.Skip* ∧ *snd* (*last l*) ∈ *Normal* ‘ *q* ∨
fst (*last l*) = *LanguageCon.com.Throw* ∧ *snd* (*last l*) ∈ *Normal* ‘ *a*
by *fastforce*
note *res=conjI*[*OF l-side r-side*]
} **thus** *?thesis* **by** *auto*
qed
qed

lemma *cpn-assum-induct*:
assumes
a0: (Γ,*l*) ∈ (*cpn n* Γ *c s*) ∧ ((Γ,*l*) ∈ *assum*(*p*, *R*)) **and**
a1: $k <$ *length l* ∧ *l*!*k*=(*c1*,*Normal s'*) ∧ $s' \in p1$
shows (Γ,*drop k l*)∈ ((*cpn n* Γ *c1* (*Normal s'*)) ∩ *assum*(*p1*, *R*))
proof −
have *drop-k-s*:(*drop k l*)!*0* = (*c1*,*Normal s'*) **using** *a1* **by** *fastforce*
have *p1*:$s' \in p1$ **using** *a1* **by** *auto*
have *k-l*:$k <$ *length l* **using** *a1* **by** *auto*

show *?thesis*
proof
show (Γ, *drop k l*) ∈ *cpn n* Γ *c1* (*Normal s′*)
unfolding *cp-def*
using *a0 a1*
by (*simp add*: *cpn-def dropcptn-is-cptn1*)
next
let *?c*= (Γ,*drop k l*)
have *l*:*snd*((*snd ?c*!*0*)) ∈ *Normal ‘ p1*
using *p1 drop-k-s* **by** *auto*
{**fix** *i*
assume *a00*:*Suc i*<*length* (*snd ?c*)
assume *a11*:(*fst ?c*)$\vdash_c$((*snd ?c*)!*i*) $\rightarrow_e$ ((*snd ?c*)!(*Suc i*))
have (*snd*((*snd ?c*)!*i*), *snd*((*snd ?c*)!(*Suc i*))) ∈ *R*
using *a0* **unfolding** *assum-def* **using** *a00 a11* **by** *auto*
} **thus** (Γ, *drop k l*) ∈ *assum* (*p1*, *R*)
using *l* **unfolding** *assum-def* **by** *fastforce*
qed
qed

lemma *cpn-comm-induct*:
assumes
a1: *l1* = *drop j l* ∧ (Γ, *l1*)∈ *comm*(*G*, (*q*,*a*)) *F* **and**
a2: *k* ≥ *j* ∧ *j* < *length l*
shows *snd* (*last* (*l*)) ∉ *Fault ‘ F* ⟶ ((*Suc k* < *length l* ⟶
Γ$\vdash_c$(*l*!*k*) → (*l*!(*Suc k*)) ⟶
(*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) ∈ *G*)
∧ (*final* (*last* (*l*)) ⟶
((*fst* (*last* (*l*)) = *Skip* ∧
snd (*last* (*l*)) ∈ *Normal ‘ q*)) ∨
(*fst* (*last* (*l*)) = *Throw* ∧
snd (*last* (*l*)) ∈ *Normal ‘* (*a*))))
proof −
have *pair-*Γ*l*:*fst* (Γ,*l1*) = Γ ∧ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
have *a03*:*snd* (*last* (*l1*)) ∉ *Fault ‘ F* ⟶(∀ *i*.
Suc i<*length* (*snd* (Γ, *l1*)) ⟶
fst (Γ, *l1*)$\vdash_c$((*snd* (Γ, *l1*))!*i*) → ((*snd* (Γ, *l1*))!(*Suc i*)) ⟶

(*snd*((*snd* (Γ, *l1*))!*i*), *snd*((*snd* (Γ, *l1*))!(*Suc i*))) ∈ *G*) ∧
(*final* (*last* (*snd* (Γ, *l1*))) ⟶
snd (*last* (*snd* (Γ, *l1*))) ∉ *Fault ‘ F* ⟶
((*fst* (*last* (*snd* (Γ, *l1*))) = *Skip* ∧
snd (*last* (*snd* (Γ, *l1*))) ∈ *Normal ‘ q*)) ∨
(*fst* (*last* (*snd* (Γ, *l1*))) = *Throw* ∧
snd (*last* (*snd* (Γ, *l1*))) ∈ *Normal ‘* (*a*)))
using *a1* **unfolding** *comm-def* **by** *fastforce*
have *last-l*:*last l1* = *last l* **using** *a1 a2* **by** *fastforce*
show *?thesis*
proof −

{
assume $snd\ (last\ l) \notin Fault\ `\ F$
then have $l1\text{-}f{:}snd\ (last\ l1) \notin Fault\ `\ F$
using $a03\ a1\ a2$ by $force$
{ assume $Suc\ k < length\ l$
then have $a2{:}\ k \geq j \wedge Suc\ k < length\ l$ using $a2$ by $auto$
have $k \leq length\ l$ using $a2$ by $fastforce$
then have $l1\text{-}l{:}(l!k = l1!\ (k - j)\ ) \wedge (l!Suc\ k = l1!Suc\ (k - j))$
using $a1\ a2$ by $fastforce$
have $a00{:}Suc\ (k - j) < length\ l1$ using $a1\ a2$ by $fastforce$
have $\Gamma \vdash_c (l1!(k-j)) \ \rightarrow (l1!(Suc\ (k-j))) \longrightarrow$
$(snd((snd\ (\Gamma,\ l1))!(k-j)),\ snd((snd\ (\Gamma,\ l1))!(Suc\ (k-j)))) \in G$
using $pair\text{-}\Gamma l\ a00\ l1\text{-}f\ a03$ by $presburger$
then have $\Gamma \vdash_c (l!k) \ \rightarrow (l!(Suc\ k)) \longrightarrow$
$(snd\ (l\ !\ k),\ snd\ (l\ !\ Suc\ k)) \in G$
using $l1\text{-}l\ last\text{-}l$ by $auto$
} then have $l\text{-}side{:}Suc\ k < length\ l \longrightarrow$
$\Gamma \vdash_c\ l\ !\ k \rightarrow l\ !\ Suc\ k \longrightarrow$
$(snd\ (l\ !\ k),\ snd\ (l\ !\ Suc\ k)) \in G$ by $auto$
{
assume $a10{:}final\ (last\ (l))$
then have $final\text{-}eq{:}\ final\ (last\ (l1))$
using $a10\ a1\ a2$ by $fastforce$
also have $snd\ (last\ (l1)) \notin Fault\ `\ F$
using $last\text{-}l\ l1\text{-}f$ by $fastforce$
ultimately have $((fst\ (last\ (snd\ (\Gamma,\ l1))) = Skip\ \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l1))) \in Normal\ `\ q)) \vee$
$(fst\ (last\ (snd\ (\Gamma,\ l1))) = Throw\ \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l1))) \in Normal\ `\ (a))$
using $pair\text{-}\Gamma l\ a03$ by $presburger$
then have $((fst\ (last\ (snd\ (\Gamma,\ l))) = Skip\ \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l))) \in Normal\ `\ q)) \vee$
$(fst\ (last\ (snd\ (\Gamma,\ l))) = Throw\ \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l))) \in Normal\ `\ (a))$
using $final\text{-}eq\ a1\ a2$ by $auto$
} then have
$r\text{-}side{:}$
$SmallStepCon.final\ (last\ l) \longrightarrow$
$fst\ (last\ l) = LanguageCon.com.Skip \wedge snd\ (last\ l) \in Normal\ `\ q\ \vee$
$fst\ (last\ l) = LanguageCon.com.Throw \wedge snd\ (last\ l) \in Normal\ `\ a$
by $fastforce$
note $res{=}conjI[OF\ l\text{-}side\ r\text{-}side]$
} thus $?thesis$ by $auto$
qed
qed

lemma $If\text{-}sound{:}$
$\Gamma,\Theta \vdash_{/F}\ c1\ sat\ [p \cap b,\ \ R,\ G,\ q,a] \Longrightarrow$

$(\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p \cap b,\ \ R,\ G,\ q,a]) \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}\ c2\ sat\ [p \cap (-b),\ \ R,\ G,\ q,a] \Longrightarrow$
$(\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [p \cap (-b),\ \ R,\ G,\ q,a]) \Longrightarrow$
$Sta\ p\ R \Longrightarrow (\forall s.\ (Normal\ s,\ Normal\ s) \in G)\ \ \Longrightarrow$
$\Gamma,\Theta \models n_{/F}\ (Cond\ b\ c1\ c2)\ sat\ [p,\ R,\ G,\ q,a]$

proof −
assume
a0:$\Gamma,\Theta \vdash_{/F}\ c1\ sat\ [p \cap b,\ \ R,\ G,\ q,a]$ **and**
a1:$\Gamma,\Theta \vdash_{/F}\ c2\ sat\ [p \cap (-b),\ R,\ G,\ q,a]$ **and**
a2: $\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p \cap b,\ R,\ G,\ q,a]$ **and**
a3: $\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [p \cap (-b),\ R,\ G,\ q,a]$ **and**
a4: *Sta p R* **and**
a5: $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$
{
fix *s*
assume *all-call*:$\forall (c,p,R,G,q,a) \in \Theta.\ \Gamma \models n_{/F}\ (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]$
then have *a3*:$\Gamma \models n_{/F}\ c2\ sat\ [p \cap (-b),\ R,\ G,\ q,a]$
using *a3 com-cvalidityn-def* **by** *fastforce*
have *a2*:$\Gamma \models n_{/F}\ c1\ sat\ [p \cap b,\ R,\ G,\ q,a]$
using *a2 all-call com-cvalidityn-def* **by** *fastforce*
have $cpn\ n\ \Gamma\ (Cond\ b\ c1\ c2)\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
proof −
{
fix *c*
assume *a10*:$c \in cpn\ n\ \Gamma\ (Cond\ b\ c1\ c2)\ s$ **and** *a11*:$c \in assum(p,\ R)$
then have *a10'*:$c \in cp\ \Gamma\ (Cond\ b\ c1\ c2)\ s$ **unfolding** *cp-def cpn-def*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
obtain *Γ1 l* **where** *c-prod*:$c=(\Gamma 1,l)$ **by** *fastforce*
have $c \in comm(G,\ (q,a))\ F$
proof −
{assume *l-f*:$snd\ (last\ l) \notin Fault\ `\ F$

have *cp*:$l!0=((Cond\ b\ c1\ c2),s) \wedge (\Gamma,l) \in cptn \wedge \Gamma=\Gamma 1$ **using** *a10′ cp-def c-prod* **by** *fastforce*
have *Γ1*:$(\Gamma,\ l) = c$ **using** *c-prod cp* **by** *blast*
have *assum*:$snd(l!0) \in Normal\ `\ (p) \wedge (\forall i.\ Suc\ i<length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:$env\text{-}tran\ \Gamma\ p\ l\ R$ **using** *env-tran-def cp* **by** *blast*
then have *env-tran-right*: $env\text{-}tran\text{-}right\ \Gamma\ l\ R$
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have *concl*:$(\forall i.\ Suc\ i<length\ l \longrightarrow$
$\Gamma 1\vdash_c(l!i)\ \ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof −
{ fix *k ns ns′*
assume *a00*:$Suc\ k<length\ l$ **and**

a21:$\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k))$
obtain *j* **where** *before-k-all-evnt*:$j\leq k \wedge (\Gamma\vdash_c(l!j) \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma\vdash_c(cj,sj) \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have *k-basic*:$cj = (Cond\ b\ c1\ c2) \wedge sj \in Normal$ ‘ (p)
using *pair-j before-k-all-evnt cp env-tran-right a4 assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain s' **where** *ss*:$sj = Normal\ s' \wedge s'\in (p)$ **by** *auto*
then have *ssj-normal-s*:$ssj = Normal\ s'$ **using** *before-k-all-evnt k-basic pair-j*
by (*metis prod.collapse snd-conv stepc-Normal-elim-cases*(*6*))
have $(snd(l!k), snd(l!(Suc\ k))) \in G$
using *ss a2* **unfolding** *Satis-def*
proof (*cases k=j*)
case *True*
have $(Normal\ s', Normal\ s') \in G$
using *a5* **by** *blast*
thus $(snd\ (l\ !\ k), snd\ (l\ !\ Suc\ k)) \in G$
using *pair-j k-basic True ss ssj-normal-s* **by** *auto*
next
case *False*
have *j-length*:$Suc\ j < length\ l$ **using** *a00 before-k-all-evnt* **by** *fastforce*
have *l-suc*:$l!(Suc\ j) = (csj, Normal\ s')$
using *before-k-all-evnt pair-j ssj-normal-s*
by *fastforce*
have *l-k*:$j<k$ **using** *before-k-all-evnt False* **by** *fastforce*
have $s'\in b \vee s'\notin b$ **by** *auto*
thus $(snd\ (l\ !\ k), snd\ (l\ !\ Suc\ k)) \in G$
proof
assume *a000*:$s'\in b$
then have *cj*:$csj=c1$ **using** *k-basic pair-j ss*
by (*metis* (*no-types*) *fst-conv stepc-Normal-elim-cases*(*6*))
moreover have *p1*:$s' \in (p \cap b)$ **using** *a000 ss* **by** *blast*
moreover have $cpn\ n\ \Gamma\ csj\ ssj \cap assum((p \cap b), R) \subseteq comm(G, (q,a))\ F$
using *calculation a2 com-validityn-def cj* **by** *blast*
ultimately have *drop-comm*:$((\Gamma, drop\ (Suc\ j)\ l))\in comm(G, (q,a))\ F$
using *l-suc j-length a10 a11* Γ*1 ssj-normal-s*
cpn-assum-induct[*of* Γ *l n* (*LanguageCon.com.Cond b c1 c2*) *s p R Suc j c1* s' $(p \cap b)$]
by *blast*
show *?thesis*
using *l-k drop-comm a00 a21 a10* Γ*1 l-f*
cpn-comm-induct
by *fastforce*

next
assume *a000*:$s' \notin b$
then have *cj*:*csj*=*c2* **using** *k-basic pair-j ss*
by (*metis* (*no-types*) *fst-conv stepc-Normal-elim-cases*(*6*))
moreover have *p1*:$s' \in (p \cap (-b))$ **using** *a000 ss* **by** *fastforce*
moreover then have *cpn n* Γ *csj ssj* $\cap$ *assum*$((p \cap (-b)), R) \subseteq$ *comm*$(G, (q,a))$ F
using *a3 com-validityn-def cj* **by** *blast*
ultimately have *drop-comm*:$((\Gamma,$ *drop* (*Suc j*) *l*$))\in$ *comm*$(G, (q,a))$ F
using *l-suc j-length a10 a11* Γ*1* *ssj-normal-s*
cpn-assum-induct[*of* Γ *l n* (*LanguageCon.com.Cond b c1 c2*) *s p R Suc j c2* s' $(p \cap (-b))$]
by *fastforce*
show *?thesis*
using *l-k drop-comm a00 a21 a10* Γ*1* *l-f*
cpn-comm-induct
unfolding *Satis-def* **by** *fastforce*
qed
qed
} **thus** *?thesis* **by** (*simp add*: *c-prod cp*) **qed**
have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ (*a*)))
proof−
{
assume *valid*:*final* (*last l*)
assume *not-fault*: *snd* (*last l*) $\notin$ *Fault* ‘ *F*
have $\exists k.\ k \geq 0 \wedge k < ((\mathit{length}\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(\mathit{Suc}\ k))) \wedge$ *final* (*l*!(*Suc k*))
proof −
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:*l*=*a1*#*l1* **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
have *final-0*:¬*final*(*l*!*0*) **using** *cp* **unfolding** *final-def* **by** *auto*
have *0*≤ (*length l*−*1*) **using** *len-l last-l* **by** *auto*
moreover have (*length l*−*1*) < *length l* **using** *len-l* **by** *auto*
moreover have *final* (*l*!(*length l*−*1*)) **using** *valid last-l* **by** *auto*
moreover have *fst* (*l*!*0*) = *LanguageCon.com.Cond b c1 c2* **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-final env-tran-right final-0*
by *blast*
qed
then obtain *k* **where** *a21*: $k \geq 0 \wedge k < ((\mathit{length}\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(\mathit{Suc}\ k))) \wedge$ *final* (*l*!(*Suc k*))

by *auto*
then have *a00*:*Suc k*<*length l* **by** *fastforce*
then obtain *j* **where** *before-k-all-evnt*:$j \leq k \wedge (\Gamma \vdash_c (l!j) \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma \vdash_c (l!k) \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma \vdash_c (cj,sj) \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have *j-length*:*Suc j* < *length l* **using** *a00 before-k-all-evnt* **by** *fastforce*

then have *k-basic*:$cj = (Cond\ b\ c1\ c2) \wedge sj \in Normal\ `\ (p)$
using *pair-j before-k-all-evnt cp env-tran-right a4 assum a00 stability*[*of p R l 0 j j* Γ]
by *fastforce*
then obtain *s'* **where** *ss*:$sj = Normal\ s' \wedge s' \in (p)$ **by** *auto*
then have *ssj-normal-s*:$ssj = Normal\ s'$ **using** *before-k-all-evnt k-basic pair-j*
by (*metis prod.collapse snd-conv stepc-Normal-elim-cases*(*6*))
have *l-suc*:$l!(Suc\ j) = (csj,\ Normal\ s')$
using *before-k-all-evnt pair-j ssj-normal-s*
by *fastforce*
have $s' \in b \vee s' \notin b$ **by** *auto*
then have $((fst\ (last\ l) = Skip\ \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw\ \wedge$
$snd\ (last\ l) \in Normal\ `\ (a))$
proof
assume *a000*:$s' \in b$
then have *cj*:*csj*=*c1* **using** *k-basic pair-j ss*
by (*metis* (*no-types*) *fst-conv stepc-Normal-elim-cases*(*6*))
moreover have *p1*:$s' \in (p \cap b)$ **using** *a000 ss* **by** *blast*
moreover then have *cpn n* Γ *csj ssj* $\cap$ *assum*($(p \cap b)$, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
using *a2 com-validityn-def cj* **by** *blast*
ultimately have *drop-comm*:((Γ, *drop* (*Suc j*) *l*))$\in$ *comm*(*G*, (*q*,*a*)) *F*
using *l-suc j-length a10 a11* Γ*1 ssj-normal-s*
cpn-assum-induct[*of* Γ *l n* (*LanguageCon.com.Cond b c1 c2*) *s p R Suc j c1 s'* ($p \cap b$)]
by *blast*
thus *?thesis*
using *j-length drop-comm a10* Γ*1 cpn-comm-induct valid not-fault*
by *blast*
next
assume *a000*:$s' \notin b$
then have *cj*:*csj*=*c2* **using** *k-basic pair-j ss*
by (*metis* (*no-types*) *fst-conv stepc-Normal-elim-cases*(*6*))
moreover have *p1*:$s' \in (p \cap (-b))$ **using** *a000 ss* **by** *blast*
moreover then have *cpn n* Γ *csj ssj* $\cap$ *assum*($(p \cap (-b))$, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
using *a3 com-validityn-def cj* **by** *blast*

ultimately have *drop-comm*:((Γ, *drop* (*Suc j*) *l*))$\in$ *comm*(*G*, (*q*,*a*)) *F*
using *l-suc j-length a10 a11* Γ*1* *ssj-normal-s*
cpn-assum-induct[*of* Γ *l n* (*LanguageCon.com.Cond b c1 c2*) *s p R Suc j c2 s'* ($p \cap (-b)$)]
by *blast*
thus *?thesis*
using *j-length drop-comm a10* Γ*1* *cpn-comm-induct valid not-fault*
by *blast*
qed
} **thus** *?thesis* **using** *l-f* **by** *fastforce* **qed**
note *res* = *conjI* [*OF concl concr*]
}
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *auto* **qed**
} **thus** *?thesis* **by** *auto* **qed**
} **thus** *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Asm-sound*:
(*c*, *p*, *R*, *G*, *q*, *a*) $\in \Theta \Longrightarrow$
Γ,Θ $\models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]

proof $-$
assume
a0:(*c*, *p*, *R*, *G*, *q*, *a*) $\in \Theta$
{ **fix** *s*
assume *all-call*:$\forall$ (*c*,*p*,*R*,*G*,*q*,*a*)$\in$ Θ. Γ $\models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]
then have Γ $\models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*] **using** *a0* **by** *auto*
} **thus** *?thesis* **unfolding** *com-cvalidityn-def* **by** *auto*
qed

lemma *events-p*:
assumes *a0*:(Γ,*cfg*#*l*)$\in$*cptn* **and**
a1:(Γ,*cfg*#*l*) $\in$ *assum* (*p*,*R*) **and**
a2:*i*<*length* (*cfg*#*l*) **and**
a3:$\forall$ *k*$\leq$*i*. *fst* ((*cfg*#*l*)!*k*) = *fst cfg* **and**
a4:*Sta p R*
shows $\exists$ *t1*. *snd*((*cfg*#*l*)!*i*)=*Normal t1* $\wedge$ *t1* $\in$ *p*
using *a2 a3*
proof(*induct i*)
case *0*
then show *?case* **using** *a1 a2* **unfolding** *assum-def* **by** *auto*
next
case (*Suc n*)
then have $\exists$ *t1*. *snd* ((*cfg* # *l*) ! *n*) = *Normal t1* $\wedge$ *t1* $\in$ *p* **by** *auto*
moreover have $\Gamma\vdash_c$ ((*cfg*#*l*)!*n*) $\rightarrow_e$ ((*cfg*#*l*)!(*Suc n*)) **using** *Suc a0*
by (*metis Env calculation less-Suc-eq-le less-not-refl nat-le-linear prod.collapse*)
then have (*snd* ((*cfg*#*l*)!*n*),*snd* ((*cfg*#*l*)!(*Suc n*)))$\in$*R* **using** *a1 Suc*(*2*)
unfolding *assum-def* **by** *auto*

ultimately show *?case* **using** *a4* **unfolding** *Sta-def* **by** *auto*
qed

lemma *not-val-zero*:$c \in dom\ \Gamma \Longrightarrow Sta\ p\ R \Longrightarrow \Gamma \models 0_{/F}\ Call\ c\ sat\ [p,R,\ G,\ q,a]$
proof−
assume *a0*: $c \in dom\ \Gamma$
assume *a1*:*Sta p R*
{**fix** *l s*
assume *a01*:$(\Gamma,l) \in cpn\ 0\ \Gamma\ (Call\ c)\ s \wedge (\Gamma,l) \in assum(p,\ R)$
then have *length* $l \geq 1$ **unfolding** *cpn-def* **using** *CptnEmpty*
by (*metis* (*no-types*, *lifting*) *One-nat-def Product-Type.Collect-case-prodD Suc-leI length-greater-0-conv snd-conv*)
moreover {**assume** *a02*:*length* $l = 1$
then have $l = [(Call\ c,s)]$
proof −
have $l\ !\ 0 = (LanguageCon.com.Call\ c,\ s)$ **using** *a01* **unfolding** *cpn-def*
by *fastforce*
then show *?thesis* **using** *a02*
by (*metis One-nat-def Suc-leI impossible-Cons length-greater-0-conv list.size*(*3*) *neq-Nil-conv nth-Cons-0 zero-neq-one*)
qed
then have $(\Gamma,l) \in comm(G,\ (q,a))\ F$ **unfolding** *comm-def final-def* **by** *auto*
}
moreover {**assume** *a02*:*length* $l > 1$
then obtain *a1 ls* **where** *l*:$l = (Call\ c,\ s)\#a1\#ls$ **using** *a01* **unfolding** *cpn-def*
apply *auto*
by (*metis* (*no-types*, *hide-lams*) *One-nat-def Suc-eq-plus1 less-not-refl list.exhaust list.size*(*3*) *list.size*(*4*) *not-less-zero nth-Cons-0 prod.collapse*)
have *l-cptn*:$(\Gamma,l) \in cptn$ **using** *a01* **unfolding** *cpn-def*
using *cptn-eq-cptn-mod-nest* **by** *blast*
then obtain *m* **where** *min-call*:*min-call m* Γ *l*
using *cptn-eq-cptn-mod-set cptn-mod-cptn-mod-nest minimum-nest-call* **by** *blast*
{ **assume** *a03*:$\forall i<length\ l.\ fst\ (l!i) = Call\ c$
then have $(\Gamma,l) \in comm(G,\ (q,a))\ F$
using *no-comp-tran-no-final-comm*[*OF - a03*] *a02* **unfolding** *final-def*
by *fastforce*
}
moreover { **assume** *a03*:$\neg\ (\forall i<length\ l.\ fst\ (l!i) = Call\ c)$
then obtain *i* **where** *i*:$(i<length\ l \wedge fst\ (l!i) \neq Call\ c)$
by *auto*
then obtain *j* **where** *cfg-j*:$fst\ (l!j) \neq Call\ c \wedge (\forall k<j.\ fst\ (l!k) = Call\ c)$

by (*fast dest*: *exists-first-occ*[*of* $\lambda i.\ fst\ (l!i) \neq Call\ c\ i$])
moreover have *j*:$j>0 \wedge j<length\ l$ **using** *l i calculation*
by (*metis gr0I fstI leI le-less-trans nth-Cons'*)
ultimately have *step*:$(\Gamma\vdash_c (l!(j-1)) \rightarrow (l!j))$

using *l l-cptn*
by (*metis One-nat-def Suc-pred cptn-stepc-rtran diff-less not-eq-not-env prod.collapse*
step-ce-elim-cases zero-less-one)
moreover obtain *s'* **where** *j-1-cfg*:*snd* (*l*!(*j*−*1*)) = *Normal s'* ∧ *s'* ∈ *p*
using *cfg-j l a01*[*simplified l*] *j*[*simplified l*] *i a1 events-p*[*OF l-cptn*[*simplified l*] - - - *a1*, *of j*−*1*]
by *force*
then have *j-cfg*:*l*!*j* = (*the* (Γ *c*), *Normal s'*) **using** *cfg-j a0*
stepc-Normal-elim-cases(*9*) *calculation*
by (*metis diff-less domIff j option.sel prod.collapse zero-less-one*)
ultimately have *False*
proof−
have (*0*,Γ, *drop* (*j*−*1*) *l*) ∈ *cptn-mod-nest-call*
using *a01* **unfolding** *cpn-def*
by (*simp add*: *dropcptn-is-cptn1 j less-imp-diff-less*)
then show *?thesis*
using *redex-call-cptn-mod-min-nest-call-gr-zero j j-cfg j-1-cfg cfg-j step a0*
by (*metis Cons-nth-drop-Suc One-nat-def SmallStepCon.redex.simps*(*7*) *Suc-pred*
diff-less domIff less-imp-diff-less min-call-def not-less-zero prod.collapse

stepc-Normal-elim-cases(*9*) *zero-less-one*)
qed
}
ultimately have (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F* **by** *auto*
}
ultimately have (Γ,*l*)∈*comm*(*G*, (*q*,*a*)) *F* **by** *fastforce*
} **then show** *?thesis* **unfolding** *com-validityn-def cpn-def* **by** *auto*
qed

lemma *Call-sound*:
f ∈ *dom* Γ ⟹
∀ *n*. Γ,Θ ⊨$n_{/F}$ (*the* (Γ *f*)) *sat* [*p*, *R*, *G*, *q*,*a*] ⟹
Sta p R ⟹ (∀ *s*. (*Normal s*,*Normal s*) ∈ *G*) ⟹
Γ,Θ ⊨$n_{/F}$ (*Call f*) *sat* [*p*, *R*, *G*, *q*,*a*]
proof −
assume
a0:*f* ∈ *dom* Γ **and**
a2:∀ *n*. Γ,Θ ⊨$n_{/F}$ (*the* (Γ *f*)) *sat* [*p*, *R*, *G*, *q*,*a*] **and**
a3: *Sta p R* **and**
a4: (∀ *s*. (*Normal s*, *Normal s*) ∈ *G*)
obtain *bdy* **where** *a0*:Γ *f* = *Some bdy* **using** *a0* **by** *auto*
{
fix *s*
assume *all-call*:∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. Γ ⊨$n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]
then have *a2*:Γ ⊨$n_{/F}$ *bdy sat* [*p*, *R*, *G*, *q*,*a*]
using *a0 a2 com-cvalidityn-def* **by** *fastforce*

have *cpn n* Γ (*Call f*) *s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*
proof –
{
fix *c*
assume *a10*:*c* ∈ *cpn n* Γ (*Call f*) *s* **and** *a11*:*c* ∈ *assum*(*p*, *R*)
then have *a10′*:*c* ∈ *cp* Γ (*Call f*) *s*
unfolding *cpn-def cp-def* **using** *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
obtain Γ*1 l* **where** *c-prod*:*c*=(Γ*1*,*l*) **by** *fastforce*
have *c* ∈ *comm*(*G*, (*q*,*a*)) *F*
proof –
{assume *l-f*:*snd* (*last l*) ∉ *Fault* ‘ *F*
have *cp*:*l*!*0*=((*Call f*),*s*) ∧ (Γ,*l*) ∈ *cptn* ∧ Γ=Γ*1* **using** *a10′ cp-def c-prod* **by** *fastforce*
have Γ*1*:(Γ, *l*) = *c* **using** *c-prod cp* **by** *blast*
have *assum*:*snd*(*l*!*0*) ∈ *Normal* ‘ (*p*) ∧ (∀ *i*. *Suc i*<*length l* ⟶
(Γ*1*)⊢$_c$(*l*!*i*) →$_e$ (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *R*)
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:*env-tran* Γ *p l R* **using** *env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have *concl*:(∀ *i*. *Suc i*<*length l* ⟶
Γ*1*⊢$_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)
proof –
{ fix *k ns ns′*
assume *a00*:*Suc k*<*length l* **and**
a21:Γ⊢$_c$(*l*!*k*) → (*l*!(*Suc k*))
obtain *j* **where** *before-k-all-evnt*:*j*≤*k* ∧ (Γ⊢$_c$(*l*!*j*) → (*l*!(*Suc j*))) ∧ (∀ *k* < *j*. (Γ⊢$_c$(*l*!*k*) →$_e$ (*l*!(*Suc k*))))
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:(Γ⊢$_c$(*cj*,*sj*) → (*csj*,*ssj*)) ∧ *cj* = *fst* (*l*!*j*) ∧ *sj* = *snd* (*l*!*j*) ∧ *csj* = *fst* (*l*!(*Suc j*)) ∧ *ssj* = *snd*(*l*!(*Suc j*))
by *fastforce*
have *k-basic*:*cj* = (*Call f*) ∧ *sj* ∈ *Normal* ‘ (*p*)
using *pair-j before-k-all-evnt cp env-tran-right a3 assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain *s′* **where** *ss*:*sj* = *Normal s′* ∧ *s′*∈ (*p*) **by** *auto*
then have *ssj-normal-s*:*ssj* = *Normal s′*
using *before-k-all-evnt k-basic pair-j a0*
by (*metis not-None-eq snd-conv stepc-Normal-elim-cases*(*9*))
have (*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) ∈ *G*
using *ss a2*
proof (*cases k*=*j*)
case *True*
have (*Normal s′*, *Normal s′*) ∈ *G*
using *a4* **by** *fastforce*

thus (*snd* (*l* ! *k*), *snd* (*l* ! *Suc k*)) $\in$ *G*
using *pair-j k-basic True ss ssj-normal-s* **by** *auto*
next
case *False*
have *j-k*:*j*<*k* **using** *before-k-all-evnt False* **by** *fastforce*
thus (*snd* (*l* ! *k*), *snd* (*l* ! *Suc k*)) $\in$ *G*
proof −
have *j-length*:*Suc j* < *length l* **using** *a00 before-k-all-evnt* **by** *fastforce*
have *cj*:*csj*=*bdy* **using** *k-basic pair-j ss a0*
by (*metis fst-conv option.distinct*(*1*) *option.sel stepc-Normal-elim-cases*(*9*))

moreover have *p1*:*s'*$\in$*p* **using** *ss* **by** *blast*
moreover then have *cpn n* Γ *csj ssj* $\cap$ *assum*(*p*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*))
F
using *a2 com-validityn-def cj* **by** *blast*
moreover then have *l*!(*Suc j*) = (*csj*, *Normal s'*)
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:((Γ, *drop* (*Suc j*) *l*))$\in$ *comm*(*G*, (*q*,*a*)) *F*
using *j-length a10 a11* Γ*1* *ssj-normal-s*
by (*meson contra-subsetD cpn-assum-induct*)
then show *?thesis*
using *a00 a21* Γ*1* *j-k j-length l-f*
cptn-comm-induct[*of* Γ *l Call f s - Suc j G q a F k*]
Suc-leI a10' **by** *blast*
qed
qed
} **thus** *?thesis* **by** (*simp add*: *c-prod cp*) **qed**
have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ (*a*)))
proof−
{
assume *valid*:*final* (*last l*)
have $\exists$ *k*. *k*$\geq$*0* $\wedge$ *k*<((*length l*) − *1*) $\wedge$ ($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*))) $\wedge$ *final*
(*l*!(*Suc k*))
proof −
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:*l*=*a1*#*l1* **by** (*metis SmallStepCon.nth-tl*
length-greater-0-conv)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
have *final-0*:¬*final*(*l*!*0*) **using** *cp* **unfolding** *final-def* **by** *auto*
have *0*$\leq$ (*length l*−*1*) **using** *len-l last-l* **by** *auto*
moreover have (*length l*−*1*) < *length l* **using** *len-l* **by** *auto*
moreover have *final* (*l*!(*length l*−*1*)) **using** *valid last-l* **by** *auto*
moreover have *fst* (*l*!*0*) = *Call f* **using** *cp* **by** *auto*

ultimately show *?thesis*
using *cp final-exist-component-tran-final env-tran-right final-0*
by *blast*
qed
then obtain k **where** *a21*: $k \geq 0 \wedge k < ((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(Suc\ k))) \wedge final\ (l!(Suc\ k))$
by *auto*
then have *a00*:$Suc\ k < length\ l$ **by** *fastforce*
then obtain j **where** *before-k-all-evnt*:$j \leq k \wedge (\Gamma \vdash_c (l!j) \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma \vdash_c (l!k) \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma \vdash_c (cj,sj) \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have $((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a))$
proof −
have *j-length*:$Suc\ j < length\ l$ **using** *a00 before-k-all-evnt* **by** *fastforce*

then have *k-basic*:$cj = (Call\ f) \wedge sj \in Normal\ `\ (p)$
using *pair-j before-k-all-evnt cp env-tran-right a3 assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain s' **where** *ss*:$sj = Normal\ s' \wedge s' \in (p)$ **by** *auto*
then have *ssj-normal-s*:$ssj = Normal\ s'$
using *before-k-all-evnt k-basic pair-j a0*
by (*metis not-None-eq snd-conv stepc-Normal-elim-cases*(*9*))
have *cj*:$csj=bdy$ **using** *k-basic pair-j ss a0*
by (*metis fst-conv option.distinct*(*1*) *option.sel stepc-Normal-elim-cases*(*9*))

moreover have *p1*:$s' \in p$ **using** *ss* **by** *blast*
moreover then have $cpn\ n\ \Gamma\ csj\ ssj \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
using *a2 com-validityn-def cj* **by** *blast*
moreover then have $l!(Suc\ j) = (csj,\ Normal\ s')$
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:$((\Gamma,\ drop\ (Suc\ j)\ l)) \in comm(G,\ (q,a))\ F$
using *j-length a10 a11* Γ*1* *ssj-normal-s*
by (*meson contra-subsetD cpn-assum-induct*)
thus *?thesis*
using *j-length l-f drop-comm a10′* Γ*1* *cptn-comm-induct*[*of* Γ *l Call f s - Suc j G q a F Suc j*] *valid*
by *blast*
qed
} **thus** *?thesis* **by** *auto*
qed

note *res* = *conjI* [*OF concl concr*]**}**
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *force* **qed**
} thus *?thesis* **by** *auto* **qed**
} thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *CallRec-sound*:
(*c*, *p*, *R*, *G*, *q*, *a*) ∈ *Specs* ⟹
∀(*c*, *p*, *R*, *G*, *q*, *a*)∈*Specs*.
c ∈ *dom* Γ ∧
Sta p R ∧ (∀*s*. (*Normal s*,*Normal s*) ∈ *G*) ∧
Γ,Θ ∪ *Specs* $\vdash_{/F}$ *the* (Γ *c*) *sat* [*p*, *R*, *G*, *q*,*a*] ∧
(∀*x*. Γ,Θ ∪ *Specs* ⊨*x* $_{/F}$ *the* (Γ *c*) *sat* [*p*,*R*, *G*, *q*,*a*]) ⟹
Sta p R ⟹ (∀*s*. (*Normal s*,*Normal s*) ∈ *G*) ⟹
Γ,Θ ⊨*n*$_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
proof −
assume *a0*: (*c*, *p*, *R*, *G*, *q*, *a*) ∈ *Specs* **and**
a1:
∀(*c*, *p*, *R*, *G*, *q*, *a*)∈*Specs*.
c ∈ *dom* Γ ∧ *Sta p R* ∧ (∀*s*. (*Normal s*,*Normal s*) ∈ *G*) ∧
Γ,Θ ∪ *Specs* $\vdash_{/F}$ *the* (Γ *c*) *sat* [*p*, *R*, *G*, *q*,*a*] ∧
(∀*x*. Γ,Θ ∪ *Specs* ⊨*x* $_{/F}$ *the* (Γ *c*) *sat* [*p*,*R*, *G*, *q*,*a*])
then have *a1'*: *c* ∈ *dom* Γ **and**
a1'': Γ,Θ ∪ *Specs* ⊨*n* $_{/F}$ *the* (Γ *c*) *sat* [*p*,*R*, *G*, *q*,*a*] **using** *a0* **by** *auto*
from *a1* **have**
valid-body:
∀(*c*, *p*, *R*, *G*, *q*, *a*)∈*Specs*.
c ∈ *dom* Γ ∧ *Sta p R* ∧ (∀*s*. (*Normal s*,*Normal s*) ∈ *G*) ∧
(∀*x*. Γ,Θ ∪ *Specs* ⊨*x* $_{/F}$ *the* (Γ *c*) *sat* [*p*,*R*, *G*, *q*,*a*]) **by** *fastforce*
assume *a5*: *Sta p R* **and**
a6: (∀*s*. (*Normal s*,*Normal s*) ∈ *G*)
obtain *bdy* **where** Γ*bdy*:Γ *c* = *Some bdy* **using** *a1'* **by** *auto*
have *theta-specs*:
∀(*c*, *p*, *R*, *G*, *q*, *a*)∈Θ. Γ ⊨*n*$_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*] ⟹
∀(*c*, *p*, *R*, *G*, *q*, *a*)∈*Specs*. Γ ⊨*n*$_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
proof(*induct n*)
case *0*
show ∀(*c*, *p*, *R*, *G*, *a*, *d*)∈*Specs*. Γ ⊨*0*$_{/F}$ *LanguageCon.com.Call c sat* [*p*,*R*,
G, *a*,*d*]
proof−
{fix *c p R G a d*
assume *a00*:(*c*, *p*, *R*, *G*, *a*, *d*) ∈ *Specs*
then have *c*∈*dom* Γ ∧ *Sta p R* **using** *a1* **by** *auto*
then have Γ ⊨*0*$_{/F}$ (*LanguageCon.com.Call c*) *sat* [*p*,*R*, *G*, *a*,*d*]
using *not-val-zero* **by** *fastforce*
} then show *?thesis* **by** *auto*
qed
next

case (*Suc n*)
have *hyp*:$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in\Theta$. $\Gamma \models n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*] $\Longrightarrow$
$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in$*Specs*. $\Gamma \models n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*] **by** *fact*
have *body*:$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in\Theta$. $\Gamma \models$*Suc* $n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*] **by** *fact*
then show *?case*
proof–
{ fix *c p R G q a*
assume *a000*:(*c*, *p*, *R*, *G*, *q*, *a*) $\in$ *Specs*
have *ctxt-m*:$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in\Theta$. $\Gamma \models n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
using *body cptn-mod-nest-mono* **unfolding** *com-validityn-def cpn-def*
by (*fastforce simp add*: *cpn-rule*)
then have *valid-Proc*:$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in$*Specs*. $\Gamma \models n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
using *hyp* **by** *auto*
have *Sta*:*Sta p R* **using** *a1 a000* **by** *auto*
have *c-dom*: *c* $\in$ *dom* Γ **using** *a1 a000* **by** *auto*
have *guar*:$\forall$ *s*. (*Normal s*,*Normal s*)$\in$*G* **using** *a1 a000* **by** *auto*
let *?*Θ'= $\Theta \cup$ *Specs*
from *valid-Proc ctxt-m*
have $\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in$*?*Θ'. $\Gamma \models n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
by *fastforce*
with *valid-body*
have *valid-body-m*:
$\forall$ (*c*, *p*, *R*, *G*, *q*, *a*)$\in$*Specs*. $\Gamma \models n_{/F}$ (*the* (Γ *c*)) *sat* [*p*,*R*, *G*, *q*,*a*]
by (*fastforce simp*:*com-cvalidityn-def*)
then have *valid-body*:$\Gamma \models n_{/F}$ (*the* (Γ *c*)) *sat* [*p*,*R*, *G*, *q*,*a*] **using** *a000* **by** *auto*
then have $\Gamma \models$*Suc* $n_{/F}$ *Call c sat* [*p*,*R*, *G*, *q*,*a*]
proof–
{ fix *l s*
assume *a01*:(Γ,*l*)$\in$*cpn* (*Suc n*) Γ (*Call c*) *s* $\wedge$ (Γ,*l*)$\in$ *assum*(*p*, *R*)
then have *length l* $\geq$ *1* **unfolding** *cpn-def* **using** *CptnEmpty*
by (*metis* (*no-types*, *lifting*) *One-nat-def Product-Type.Collect-case-prodD Suc-leI length-greater-0-conv snd-conv*)
moreover {
assume *a02*:*length l* = *1*
then have *l* = [(*Call c*,*s*)]
proof –
have *l* ! *0* = (*LanguageCon.com.Call c*, *s*) **using** *a01* **unfolding** *cpn-def*
by *fastforce*
then show *?thesis* **using** *a02*
by (*metis One-nat-def Suc-leI impossible-Cons length-greater-0-conv list.size*(*3*) *neq-Nil-conv nth-Cons-0 zero-neq-one*)
qed
then have (Γ,*l*) $\in$ *comm*(*G*, (*q*,*a*)) *F* **unfolding** *comm-def final-def* **by**

auto
}
moreover {**assume** *a02*:*length l* > *1*
then obtain *a1 ls* **where** *l*:*l* =(*Call c*, *s*)#*a1*#*ls* **using** *a01* **unfolding** *cpn-def*
apply *auto*
by (*metis* (*no-types*, *hide-lams*) *One-nat-def Suc-eq-plus1 less-not-refl list.exhaust list.size*(*3*) *list.size*(*4*) *not-less-zero nth-Cons-0 prod.collapse*)
have *l-cptn*:(Γ,*l*)∈*cptn* **using** *a01* **unfolding** *cpn-def*
using *cptn-eq-cptn-mod-nest* **by** *blast*
then obtain *m* **where** *min-call*:*min-call m* Γ *l*
using *cptn-eq-cptn-mod-set cptn-mod-cptn-mod-nest minimum-nest-call* **by** *blast*
{ **assume** *a03*:∀ *i*<*length l*. *fst* (*l*!*i*) = *Call c*
then have (Γ,*l*) ∈ *comm*(*G*, (*q*,*a*)) *F*
using *no-comp-tran-no-final-comm*[*OF* - *a03*] *a02* **unfolding** *final-def*
by *fastforce*
}
moreover{
assume *a03*:¬(∀ *i*<*length l*. *fst* (*l*!*i*) = *Call c*)
then obtain *i* **where** *i*:(*i*<*length l* ∧ *fst* (*l*!*i*) ≠ *Call c*)
by *auto*
then obtain *j* **where** *cfg-j*:*fst* (*l*!*j*) ≠ *Call c* ∧ (∀ *k*<*j*. *fst* (*l*!*k*) = *Call c*)
by (*fast dest*: *exists-first-occ*[*of* λ*i*. *fst* (*l*!*i*) ≠ *Call c i*])
moreover have *j*:*j*>*0* ∧ *j*<*length l* **using** *l i calculation*
by (*metis gr0I fstI leI le-less-trans nth-Cons'*)
ultimately have *step*:(Γ⊢$_c$ (*l*!(*j*−*1*)) → (*l*!*j*))
using *l l-cptn*
by (*metis One-nat-def Suc-pred cptn-stepc-rtran diff-less not-eq-not-env prod.collapse*
step-ce-elim-cases zero-less-one)
then obtain *s'* **where** *j-1-cfg*:*snd* (*l*!(*j*−*1*)) = *Normal s'* ∧ *s'* ∈ *p*
using *cfg-j l a01*[*simplified l*] *j*[*simplified l*] *i Sta events-p*[*OF l-cptn*[*simplified l*] - - - *Sta*, *of j*−*1*]
by *force*
then have *j-cfg*:*l*!*j* = (*the* (Γ *c*), *Normal s'*)
using *cfg-j c-dom stepc-Normal-elim-cases*(*9*) *step*
by (*metis diff-less domIff j option.sel prod.collapse zero-less-one*)
then have *suc-n-call*:(*Suc n*,Γ, *drop* (*j*−*1*) *l*) ∈ *cptn-mod-nest-call*
using *a01* **unfolding** *cpn-def*
by (*simp add*: *dropcptn-is-cptn1 j less-imp-diff-less*)
have (*n*,Γ, *drop j l*) ∈ *cptn-mod-nest-call*
proof−
have ¬ (Γ⊢$_c$ (*l*!(*j*−*1*)) →$_e$ (*l*!*j*)) **using** *step*
by (*metis etranE mod-env-not-component*)
then have (*Suc n*,Γ, (*Call c*, *Normal s'*)#(*the* (Γ *c*),*Normal s'*)#(*drop* (*j*+*1*) *l*)) ∈ *cptn-mod-nest-call*
using *a01 j step cfg-j j-cfg j-1-cfg suc-n-call*

by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-eq-plus1*
Suc-less-eq Suc-pred diff-less less-SucI prod.collapse zero-less-one)

then have (n,Γ, (*the* (Γ *c*),*Normal s'*)#(*drop* (j+*1*) l)) ∈ *cptn-mod-nest-call*
using *cfg-j j-cfg elim-cptn-mod-nest-call-n-dec*[*OF* -] *c-dom* **by** *fastforce*
then show *?thesis*
by (*metis Cons-nth-drop-Suc Suc-eq-plus1 j j-cfg*)
qed
moreover have (Γ, *drop j l*) ∈ *assum*(p,R)
proof−
have (Γ, *take j l* @ *l* ! *j* # *drop* (*Suc j*) *l*) ∈ *assum* (p, R)
using *conjunct2*[*OF a01*] *id-take-nth-drop*[*OF conjunct2*[*OF j*]] **by** *auto*
then show *?thesis*
using *sub-assum-r*[*OF*] *j-1-cfg l j-cfg j*
by (*metis Cons-nth-drop-Suc image-eqI snd-conv*)
qed
ultimately have *comm-drop*:(Γ, *drop j l*)∈ *comm*(G, (q,a)) F
using *valid-body j-cfg j* **unfolding** *com-validityn-def cpn-def*
by *fastforce*
have (Γ,l) ∈ *comm*(G, (q,a)) F
proof−
have *h*:∀ *j*<*length* (*take j l*). *fst* ((*take j l*)!*j*) = (*Call c*) **using** *j cfg-j* **by** *fastforce*
then have *comm-take*:(Γ,*take j l*) ∈ *comm*(G, (q,a)) F
using *no-comp-tran-no-final-comm*[*of take j l Call c*] *j-1-cfg l j-cfg j cfg-j*
unfolding *final-def* **by** *auto*
moreover have (*snd* (*last* (*take j l*)), *snd* (*drop j l* ! *0*)) ∈ G
proof−
have *length* (*take j l*) = *j* **using** *l j-1-cfg j j-cfg* **by** *auto*
moreover have (*take j l*)!(j−*1*) = *l*!(j−*1*)
using *l j-1-cfg j j-cfg* **by** *auto*
ultimately have *last* (*take j l*) = *l*!(j−*1*)
using *j* **by** (*metis last-conv-nth less-numeral-extra*(*3*) *list.size*(*3*))

then show *?thesis* **using** *l j-1-cfg j j-cfg guar* **by** *auto*
qed
ultimately show *?thesis* **using** *j-1-cfg j-cfg j cfg-j j l-cptn*
comm-union[*OF comm-take comm-drop*] **by** *fastforce*
qed
} **ultimately have** (Γ,l)∈*comm*(G, (q,a)) F **by** *auto*
}ultimately have (Γ,l)∈*comm*(G, (q,a)) F **by** *fastforce*
} **thus** *?thesis* **unfolding** *com-validityn-def* **using** *cpn-rule2* **by** *blast*
qed
} **thus** *?case* **by** *fastforce*

qed
qed
then show *?thesis* **using** *a0* **unfolding** *com-cvalidityn-def* **by** *auto*
qed

lemma *Seq-env-P*:**assumes** $a0{:}\Gamma\vdash_c(Seq\ P\ Q,s) \rightarrow_e (Seq\ P\ Q,t)$
shows $\Gamma\vdash_c(P,s) \rightarrow_e (P,t)$
using *a0*
by (*metis env-not-normal-s snormal-enviroment*)

lemma *map-eq-state*:
assumes
$a0{:}(\Gamma,l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s)$ **and**
$a1{:}(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
$a2{:}l1=map\ (lift\ c2)\ l2$
shows
$\forall i<length\ l1.\ snd\ (l1!i) = snd\ (l2!i)$
using *a0 a1 a2* **unfolding** *cp-def*
by (*simp add*: *snd-lift*)

lemma *map-eq-seq-c*:
assumes
$a0{:}(\Gamma,l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s)$ **and**
$a1{:}(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
$a2{:}l1=map\ (lift\ c2)\ l2$
shows
$\forall i<length\ l1.\ fst\ (l1!i) = Seq\ (fst\ (l2!i))\ c2$
proof −
{**fix** *i*
assume $a3{:}i<length\ l1$
have $fst\ (l1!i) = Seq\ (fst\ (l2!i))\ c2$
using *a0 a1 a2 a3* **unfolding** *lift-def*
by (*simp add*: *case-prod-unfold*)
}**thus** *?thesis* **by** *auto*
qed

lemma *same-env-seq-c*:
assumes
$a0{:}(\Gamma,l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s)$ **and**
$a1{:}(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
$a2{:}l1=map\ (lift\ c2)\ l2$
shows
$\forall i.\ Suc\ i<length\ l2 \longrightarrow \Gamma\vdash_c(l2!i) \rightarrow_e (l2!(Suc\ i)) =$
$\Gamma\vdash_c(l1!i) \rightarrow_e (l1!(Suc\ i))$
proof −
have $a0a{:}(\Gamma,l1) \in cptn \wedge l1!0 = ((Seq\ c1\ c2),s)$
using *a0* **unfolding** *cp-def* **by** *blast*
have $a1a{:}\ (\Gamma,l2) \in cptn \wedge l2!0 = (c1,s)$
using *a1* **unfolding** *cp-def* **by** *blast*

{
fix *i*
assume *a3*:*Suc i*< *length l2*
have $\Gamma\vdash_c(l2!i) \ \rightarrow_e \ (l2!(Suc\ i)) =$
$\Gamma\vdash_c(l1!i) \ \rightarrow_e \ (l1!(Suc\ i))$
proof
{
assume *a4*:$\Gamma\vdash_c$ *l2* ! *i* $\rightarrow_e$ *l2* ! *Suc i*
obtain *c1i s1i c1si s1si* **where** *l1prod*:*l1* ! *i*=(*c1i*,*s1i*) ∧ *l1*!*Suc i* = (*c1si*,*s1si*)
by *fastforce*
obtain *c2i s2i c2si s2si* **where** *l2prod*:*l2* ! *i*=(*c2i*,*s2i*) ∧ *l2*!*Suc i* = (*c2si*,*s2si*)
by *fastforce*
then have *c1i* = (*Seq c2i c2*) ∧ *c1si* = (*Seq c2si c2*)
using *a0 a1 a2 a3 a4 map-eq-seq-c l1prod*
by (*metis Suc-lessD fst-conv length-map*)
also have *s2i*=*s1i* ∧ *s2si*=*s1si*
using *a0 a1 a4 a2 a3 l2prod map-eq-state l1prod*
by (*metis Suc-lessD nth-map snd-conv snd-lift*)
ultimately show $\Gamma\vdash_c$ *l1* ! *i* $\rightarrow_e$ (*l1* ! *Suc i*)
using *a4 l1prod l2prod*
by (*metis Env-n env-c-c′ env-not-normal-s step-e.Env*)
}
{
assume *a4*:$\Gamma\vdash_c$ *l1* ! *i* $\rightarrow_e$ *l1* ! *Suc i*
obtain *c1i s1i c1si s1si* **where** *l1prod*:*l1* ! *i*=(*c1i*,*s1i*) ∧ *l1*!*Suc i* = (*c1si*,*s1si*)
by *fastforce*
obtain *c2i s2i c2si s2si* **where** *l2prod*:*l2* ! *i*=(*c2i*,*s2i*) ∧ *l2*!*Suc i* = (*c2si*,*s2si*)
by *fastforce*
then have *c1i* = (*Seq c2i c2*) ∧ *c1si* = (*Seq c2si c2*)
using *a0 a1 a2 a3 a4 map-eq-seq-c l1prod*
by (*metis Suc-lessD fst-conv length-map*)
also have *s2i*=*s1i* ∧ *s2si*=*s1si*
using *a0 a1 a4 a2 a3 l2prod map-eq-state l1prod*
by (*metis Suc-lessD nth-map snd-conv snd-lift*)
ultimately show $\Gamma\vdash_c$ *l2* ! *i* $\rightarrow_e$ (*l2* ! *Suc i*)
using *a4 l1prod l2prod*
by (*metis Env-n LanguageCon.com.inject*(*3*) *env-c-c′ env-not-normal-s step-e.Env*)
}
qed
}
thus *?thesis* **by** *auto*
qed

lemma *same-comp-seq-c*:
assumes
a0:(Γ,*l1*) ∈ (*cp* Γ (*Seq c1 c2*) *s*) **and**

```
  a1:(Γ,l2) ∈ (cp Γ c1 s)  and
  a2:l1=map (lift c2) l2
shows
∀ i. Suc i<length l2 ⟶ Γ⊢c(l2!i)  → (l2!(Suc i)) =
            Γ⊢c(l1!i)  → (l1!(Suc i))
proof –
  have a0a:(Γ,l1) ∈cptn ∧ l1!0 = ((Seq c1 c2),s)
    using a0 unfolding cp-def by blast
  have a1a: (Γ,l2) ∈cptn ∧ l2!0 = (c1,s)
    using a1 unfolding cp-def by blast
  {
    fix i
    assume a3:Suc i< length l2
    have Γ⊢c(l2!i)  → (l2!(Suc i)) =
            Γ⊢c(l1!i)  → (l1!(Suc i))
    proof
    {
      assume a4:Γ⊢c l2 ! i → l2 ! Suc i
     obtain c1i s1i c1si s1si where l1prod:l1 ! i=(c1i,s1i) ∧ l1!Suc i = (c1si,s1si)
        by fastforce
     obtain c2i s2i c2si s2si where l2prod:l2 ! i=(c2i,s2i) ∧ l2!Suc i = (c2si,s2si)
        by fastforce
      then have c1i = (Seq c2i c2) ∧ c1si = (Seq c2si c2)
        using  a0 a1 a2 a3 a4  map-eq-seq-c l1prod
        by (metis Suc-lessD  fst-conv length-map)
      also have s2i=s1i ∧ s2si=s1si
        using  a0 a1 a4  a2 a3 l2prod map-eq-state l1prod
        by (metis Suc-lessD  nth-map snd-conv snd-lift)
      ultimately show Γ⊢c l1 ! i → (l1 ! Suc i)
        using a4 l1prod l2prod
        by (simp add: Seqc)
    }
    {
      assume a4:Γ⊢c l1 ! i → l1 ! Suc i
     obtain c1i s1i c1si s1si where l1prod:l1 ! i=(c1i,s1i) ∧ l1!Suc i = (c1si,s1si)
        by fastforce
     obtain c2i s2i c2si s2si where l2prod:l2 ! i=(c2i,s2i) ∧ l2!Suc i = (c2si,s2si)
        by fastforce
      then have c1i = (Seq c2i c2) ∧ c1si = (Seq c2si c2)
        using  a0 a1 a2 a3 a4  map-eq-seq-c l1prod
        by (metis Suc-lessD  fst-conv length-map)
      also have s2i=s1i ∧ s2si=s1si
        using  a0 a1 a4  a2 a3 l2prod map-eq-state l1prod
        by (metis Suc-lessD  nth-map snd-conv snd-lift)
      ultimately show Γ⊢c l2 ! i → (l2 ! Suc i)
        using a4 l1prod l2prod stepc-elim-cases-Seq-Seq
      by auto
    }
    qed
```

}
thus *?thesis* **by** *auto*
qed

lemma *assum-map*:
assumes
a0:$(\Gamma, l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s) \wedge ((\Gamma, l1) \in assum(p,\ R))$ **and**
a1:$(\Gamma, l2) \in (cp\ \Gamma\ c1\ s)$ **and**
a2:$l1 = map\ (lift\ c2)\ l2$
shows
$((\Gamma, l2) \in assum(p,\ R))$
proof −
have *a3*: $\forall i.\ Suc\ i < length\ l2 \longrightarrow \Gamma \vdash_c (l2!i) \rightarrow_e (l2!(Suc\ i)) = \Gamma \vdash_c (l1!i) \rightarrow_e (l1!(Suc\ i))$
using *a0 a1 a2 same-env-seq-c* **by** *fastforce*
have *pair-Γl1*:$fst\ (\Gamma, l1) = \Gamma \wedge snd\ (\Gamma, l1) = l1$ **by** *fastforce*
have *pair-Γl2*:$fst\ (\Gamma, l2) = \Gamma \wedge snd\ (\Gamma, l2) = l2$ **by** *fastforce*
have *drop-k-s*:$l2!0 = (c1, s)$ **using** *a1 cp-def* **by** *blast*
have *eq-length*:$length\ l1 = length\ l2$ **using** *a2* **by** *auto*
obtain s' **where** *normal-s*:$s = Normal\ s'$
using *a0* **unfolding** *cp-def assum-def* **by** *fastforce*
then have *p1*:$s' \in p$ **using** *a0* **unfolding** *cp-def assum-def* **by** *fastforce*
show *?thesis*
proof −
let $?c = (\Gamma, l2)$
have *l*:$snd((snd\ ?c!0)) \in Normal\ `\ (p)$
using *p1 drop-k-s a1 normal-s* **unfolding** *cp-def* **by** *auto*
{**fix** *i*
assume *a00*:$Suc\ i < length\ (snd\ ?c)$
assume *a11*:$(fst\ ?c) \vdash_c ((snd\ ?c)!i) \rightarrow_e ((snd\ ?c)!(Suc\ i))$
have $(snd((snd\ ?c)!i),\ snd((snd\ ?c)!(Suc\ i))) \in R$
using *a0 a1 a2 a3 map-eq-state* **unfolding** *assum-def*
using *a00 a11 eq-length* **by** *fastforce*
} **thus** $(\Gamma,\ l2) \in assum\ (p,\ R)$
using *l* **unfolding** *assum-def* **by** *fastforce*
qed
qed

lemma *comm-map'*:
assumes
a0:$(\Gamma, l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s)$ **and**
a1:$(\Gamma, l2) \in (cp\ \Gamma\ c1\ s) \wedge (\Gamma,\ l2) \in comm(G,\ (q, a))\ F$ **and**
a2:$l1 = map\ (lift\ c2)\ l2$
shows
$snd\ (last\ l1) \notin Fault\ `\ F \longrightarrow (Suc\ k < length\ l1 \longrightarrow$
$\Gamma \vdash_c (l1!k) \rightarrow (l1!(Suc\ k)) \longrightarrow$
$(snd(l1!k),\ snd(l1!(Suc\ k))) \in G) \wedge$
$(fst\ (last\ l1) = (Seq\ c\ c2) \wedge final\ (c,\ snd\ (last\ l1)) \longrightarrow$
$(fst\ (last\ l1) = (Seq\ Skip\ c2) \wedge$

$(snd\ (last\ \ l1) \in Normal\ `\ q) \vee$
$(fst\ (last\ l1) = (Seq\ Throw\ c2) \wedge$
$snd\ (last\ l1) \in Normal\ `\ (a))))$

proof –
have $a3{:}\forall\, i.\ Suc\ i{<}length\ l2 \longrightarrow \Gamma\vdash_c(l2!i)\ \ \rightarrow (l2!(Suc\ i)) =$
$\Gamma\vdash_c(l1!i)\ \ \rightarrow (l1!(Suc\ i))$
using *a0 a1 a2 same-comp-seq-c*
by *fastforce*
have $pair\text{-}\Gamma l1{:}fst\ (\Gamma,l1) = \Gamma \wedge snd\ (\Gamma,l1) = l1$ **by** *fastforce*
have $pair\text{-}\Gamma l2{:}fst\ (\Gamma,l2) = \Gamma \wedge snd\ (\Gamma,l2) = l2$ **by** *fastforce*
have $drop\text{-}k\text{-}s{:}l2!0 = (c1,s)$ **using** *a1 cp-def* **by** *blast*
have $eq\text{-}length{:}length\ l1 = length\ l2$ **using** *a2* **by** *auto*
then have $len0{:}length\ l1{>}0$ **using** *a0* **unfolding** *cp-def*
using *Collect-case-prodD drop-k-s eq-length* **by** *auto*
then have $l1\text{-}not\text{-}empty{:}l1{\neq}[]$ **by** *auto*
then have $l2\text{-}not\text{-}empty{:}l2 \neq []$ **using** *a2* **by** *blast*
have $last\text{-}lenl1{:}last\ l1 = l1!((length\ l1) - 1)$
using *last-conv-nth l1-not-empty* **by** *auto*
have $last\text{-}lenl2{:}last\ l2 = l2!((length\ l2) - 1)$
using *last-conv-nth l2-not-empty* **by** *auto*
have $a03{:}snd\ (last\ l2) \notin Fault\ `\ F\ \ \longrightarrow(\forall\, i\ ns\ ns'.$
$Suc\ i{<}length\ (snd\ (\Gamma,\ l2)) \longrightarrow$
$fst\ \ (\Gamma,\ \ l2)\vdash_c((snd\ \ (\Gamma,\ \ l2))!i)\ \ \ \rightarrow\ ((snd\ \ (\Gamma,\ \ l2))!(Suc\ \ i))\ \ \longrightarrow$

$(snd((snd\ (\Gamma,\ l2))!i),\ snd((snd\ (\Gamma,\ l2))!(Suc\ i))) \in G) \wedge$
$(final\ (last\ (snd\ (\Gamma,\ l2)))\ \ \longrightarrow$
$((fst\ (last\ (snd\ (\Gamma,\ l2))) = Skip \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l2))) \in Normal\ `\ q)) \vee$
$(fst\ (last\ (snd\ (\Gamma,\ l2))) = Throw \wedge$
$snd\ (last\ (snd\ (\Gamma,\ l2))) \in Normal\ `\ (a)))$
using *a1* **unfolding** *comm-def* **by** *fastforce*
show *?thesis* **unfolding** *comm-def*
proof –
{ fix $k\ ns\ ns'$
assume $a00a{:}snd\ (last\ l1) \notin Fault\ `\ F$
assume $a00{:}Suc\ k < length\ l1$
then have $k \leq length\ l1$ **using** *a2* **by** *fastforce*
have $a00{:}Suc\ k < length\ l2$ **using** *eq-length a00* **by** *fastforce*
then have $a00a{:}snd\ (last\ l2) \notin Fault\ `\ F$
proof–
have $snd\ (l1!((length\ l1) - 1)) = snd\ (l2!((length\ l2) - 1))$
using *a2 a1 a0 map-eq-state eq-length l2-not-empty last-snd*
by *fastforce*
then have $snd(last\ l2) = snd\ (last\ l1)$
using *last-lenl1 last-lenl2* **by** *auto*
thus *?thesis* **using** *a00a* **by** *auto*
qed
then have $snd\ (last\ l1) \notin Fault\ `\ F \longrightarrow\Gamma\vdash_c(l1!k)\ \ \rightarrow (l1!(Suc\ k)) \longrightarrow$

$(snd((snd\ (\Gamma,\ l1))!k),\ snd((snd\ (\Gamma,\ l1))!(Suc\ k))) \in\ G$
using *pair-Γl1 pair-Γl2 a00 a03 a3 eq-length a00a*
by (*metis Suc-lessD a0 a1 a2 map-eq-state*)
} **note** *l=this*
{
assume *a00*: $fst\ (last\ l1) = (Seq\ c\ c2) \wedge final\ (c,\ snd\ (last\ l1))$ **and**
a01:$snd\ (last\ (l1)) \notin Fault\ \text{‘}\ F$
then have $c{:}c{=}Skip \vee c = Throw$
unfolding *final-def* **by** *auto*
then have *fst-last-l2*:$fst\ (last\ l2) = c$
using *last-lenl1 a00 l1-not-empty eq-length len0 a2 last-conv-nth last-lift*
by *fastforce*
also have *last-eq*:$snd\ (last\ l2) = snd\ (last\ l1)$
using *l2-not-empty a2 last-conv-nth last-lenl1 last-snd*
by *fastforce*
ultimately have $final\ (fst\ (last\ l2), snd\ (last\ l2))$
using *a00* **by** *auto*
then have $final\ (last\ l2)$ **by** *auto*
also have $snd\ (last\ (l2)) \notin Fault\ \text{‘}\ F$
using *last-eq a01* **by** *auto*
ultimately have $(fst\ (last\ \ l2)) = Skip\ \wedge$
$snd\ (last\ \ l2) \in Normal\ \text{‘}\ q \vee$
$(fst\ (last\ l2) = Throw\ \wedge$
$snd\ (last\ l2) \in Normal\ \text{‘}\ (a))$
using *a03* **by** *auto*
then have $(fst\ (last\ l1) = (Seq\ Skip\ c2)\ \wedge$
$snd\ (last\ \ l1) \in Normal\ \text{‘}\ q) \vee$
$(fst\ (last\ l1) = (Seq\ Throw\ c2)\ \wedge$
$snd\ (last\ l1) \in Normal\ \text{‘}\ (a))$
using *last-eq fst-last-l2 a00* **by** *force*
}
thus *?thesis* **using** *l* **by** *auto* **qed**
qed

lemma *comm-map''*:
assumes
a0:$(\Gamma,l1) \in (cp\ \Gamma\ (Seq\ c1\ c2)\ s)$ **and**
a1:$(\Gamma,l2) \in (cp\ \Gamma\ c1\ s) \wedge (\Gamma,\ l2) \in comm(G,\ (q,a))\ F$ **and**
a2:$l1{=}map\ (lift\ c2)\ l2$
shows
$snd\ (last\ l1) \notin Fault\ \text{‘}\ F \longrightarrow ((Suc\ k < length\ l1 \longrightarrow$
$\Gamma\vdash_c(l1!k)\ \rightarrow (l1!(Suc\ k)) \longrightarrow$
$(snd(l1!k),\ snd(l1!(Suc\ k))) \in G) \wedge$
$(final\ (last\ l1) \longrightarrow$
$(fst\ (last\ l1) = Skip\ \wedge$
$(snd\ (last\ \ l1) \in Normal\ \text{‘}\ r) \vee$
$(fst\ (last\ l1) = Throw\ \wedge$
$snd\ (last\ l1) \in Normal\ \text{‘}\ (a)))))$

proof −
  have *a3*:$\forall$ *i*. *Suc i*<*length l2* $\longrightarrow$ $\Gamma\vdash_c(l2!i) \ \rightarrow (l2!(Suc\ i))$ =
    $\Gamma\vdash_c(l1!i) \ \rightarrow (l1!(Suc\ i))$
   using *a0 a1 a2 same-comp-seq-c*
   by *fastforce*
  have *pair-*Γ*l1*:*fst* (Γ,*l1*) = Γ $\wedge$ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
  have *pair-*Γ*l2*:*fst* (Γ,*l2*) = Γ $\wedge$ *snd* (Γ,*l2*) = *l2* **by** *fastforce*
  have *drop-k-s*:*l2*!*0* = (*c1*,*s*) **using** *a1 cp-def* **by** *blast*
  have *eq-length*:*length l1* = *length l2* **using** *a2* **by** *auto*
  then have *len0*:*length l1*>*0* **using** *a0* **unfolding** *cp-def*
   using *Collect-case-prodD drop-k-s eq-length* **by** *auto*
  then have *l1-not-empty*:*l1*$\neq$[] **by** *auto*
  then have *l2-not-empty*:*l2* $\neq$ [] **using** *a2* **by** *blast*
  have *last-lenl1*:*last l1* = *l1*!((*length l1*) −*1*)
     using *last-conv-nth l1-not-empty* **by** *auto*
  have *last-lenl2*:*last l2* = *l2*!((*length l2*) −*1*)
     using *last-conv-nth l2-not-empty* **by** *auto*
  have *a03*:*snd* (*last l2*) $\notin$ *Fault* ‘ *F* $\longrightarrow$($\forall$ *i ns ns'*.
      Suc i<*length* (*snd* (Γ, *l2*)) $\longrightarrow$
        fst (Γ, *l2*)$\vdash_c$((*snd* (Γ, *l2*))!*i*) $\rightarrow$ ((*snd* (Γ, *l2*))!(*Suc i*)) $\longrightarrow$

       (*snd*((*snd* (Γ, *l2*))!*i*), *snd*((*snd* (Γ, *l2*))!(*Suc i*))) $\in$ *G*) $\wedge$
      (*final* (*last* (*snd* (Γ, *l2*))) $\longrightarrow$
       ((*fst* (*last* (*snd* (Γ, *l2*))) = *Skip* $\wedge$
        snd (*last* (*snd* (Γ, *l2*))) $\in$ *Normal* ‘ *q*)) $\vee$
       (*fst* (*last* (*snd* (Γ, *l2*))) = *Throw* $\wedge$
        snd (*last* (*snd* (Γ, *l2*))) $\in$ *Normal* ‘ (*a*)))
  using *a1* **unfolding** *comm-def* **by** *fastforce*
  show *?thesis* **unfolding** *comm-def*
  proof −
  { **fix** *k ns ns'*
   assume *a00a*:*snd* (*last l1*) $\notin$ *Fault* ‘ *F*
   assume *a00*:*Suc k* < *length l1*
   then have *k* $\leq$ *length l1* **using** *a2* **by** *fastforce*
   have *a00*:*Suc k* < *length l2* **using** *eq-length a00* **by** *fastforce*
   then have *a00a*:*snd* (*last l2*) $\notin$ *Fault* ‘ *F*
   proof−
    have *snd* (*l1*!((*length l1*) −*1*)) = *snd* (*l2*!((*length l2*) −*1*))
     using *a2 a1 a0 map-eq-state eq-length l2-not-empty last-snd*
     by *fastforce*
    then have *snd*(*last l2*) = *snd* (*last l1*)
     using *last-lenl1 last-lenl2* **by** *auto*
    thus *?thesis* **using** *a00a* **by** *auto*
   qed
   then have $\Gamma\vdash_c(l1!k) \ \rightarrow (l1!(Suc\ k))$ $\longrightarrow$
     (*snd*((*snd* (Γ, *l1*))!*k*), *snd*((*snd* (Γ, *l1*))!(*Suc k*))) $\in$ *G*
    using *pair-*Γ*l1 pair-*Γ*l2 a00 a03 a3 eq-length a00a*
    by (*metis* (*no-types*,*lifting*) *a2 Suc-lessD nth-map snd-lift*)
  } **note** *l*= *this*

{
assume *a00*: *final* (*last l1*)
then have *c*:*fst* (*last l1*)=*Skip* ∨ *fst* (*last l1*) = *Throw*
unfolding *final-def* **by** *auto*
moreover have *fst* (*last l1*) = *Seq* (*fst* (*last l2*)) *c2*
using *a2 last-lenl1 eq-length*
proof −
have *last l2* = *l2* ! (*length l2* − *1*)
using *l2-not-empty last-conv-nth* **by** *blast*
then show *?thesis*
by (*metis One-nat-def a2 l2-not-empty last-lenl1 last-lift*)
qed
ultimately have *False* **by** *simp*
} **thus** *?thesis* **using** *l* **by** *auto* **qed**
qed

lemma *comm-map*:
assumes
a0:(Γ,*l1*) ∈ (*cp* Γ (*Seq c1 c2*) *s*) **and**
a1:(Γ,*l2*) ∈ (*cp* Γ *c1 s*) ∧ (Γ, *l2*)∈ *comm*(*G*, (*q*,*a*)) *F* **and**
a2:*l1*=*map* (*lift c2*) *l2*
shows
(Γ, *l1*)∈ *comm*(*G*, (*r*,*a*)) *F*
proof −
{**fix** *i*
have *snd* (*last l1*) ∉ *Fault* ‘ *F* ⟶(*Suc i* < *length* (*l1*) ⟶
$\Gamma\vdash_c$ (*l1* ! *i*) → (*l1* ! (*Suc i*)) ⟶
(*snd* (*l1* ! *i*), *snd* (*l1* ! *Suc i*)) ∈ *G*) ∧
(*SmallStepCon.final* (*last l1*) ⟶
fst (*last l1*) = *LanguageCon.com.Skip* ∧
snd (*last l1*) ∈ *Normal* ‘ *r* ∨
fst (*last l1*) = *LanguageCon.com.Throw* ∧
snd (*last l1*) ∈ *Normal* ‘ *a*)
using *comm-map′′*[*of* Γ *l1 c1 c2 s l2 G q a F i r*] *a0 a1 a2*
by *fastforce*
} **then show** *?thesis* **using** *comm-def* **unfolding** *comm-def* **by** *force*
qed

lemma *Seq-sound1*:
assumes
a0:(*n*,Γ,*x*)∈*cptn-mod-nest-call* **and**
a1:*x*!*0* = ((*Seq P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:¬ *final* (*last x*) **and**
a4:*env-tran-right* Γ *x rely* **and**
a5:*snd* (*x*!*0*)∈ *Normal* ‘ *p* ∧ *Sta p rely* ∧ *Sta a rely* **and**
a6: $\Gamma \models n_{/F}$ *P sat* [*p*, *rely*, *G*, *q*,*a*]

shows
$\exists$ *xs*. (Γ,*xs*) $\in$ *cpn n* Γ *P s* $\wedge$ *x* = *map* (*lift Q*) *xs*
using *a0 a1 a2 a3 a4 a5 a6*
proof (*induct arbitrary*: *P s p*)
case (*CptnModNestOne n* Γ *C s1*)
then have (Γ, [(*P*,*s*)]) $\in$ *cpn n* Γ *P s* $\wedge$ [(*C*, *s1*)] = *map* (*lift Q*) [(*P*,*s*)]
unfolding *cpn-def lift-def*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
then have *C*:*C*=*Seq P Q* **unfolding** *lift-def* **by** *fastforce*
have $\exists$ *xs*. (Γ, *xs*) $\in$ *cpn n* Γ *P t1* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*
proof −
have ((*C*, *t1*) # *xsa*) ! *0* = (*LanguageCon.com.Seq P Q*, *t1*) **using** *C* **by** *auto*
moreover have $\forall$ *i*<*length* ((*C*, *t1*) # *xsa*). *fst* (((*C*, *t1*) # *xsa*) ! *i*) $\neq$ *Q*
using *CptnModNestEnv*(*5*) **by** *fastforce*
moreover have $\neg$ *SmallStepCon.final* (*last* ((*C*, *t1*) # *xsa*)) **using** *CptnModNestEnv*(*6*)
by *fastforce*
moreover have *snd* (((*C*, *t1*) # *xsa*) ! *0*) $\in$ *Normal* ‘ *p*
using *CptnModNestEnv*(*8*) *CptnModNestEnv*(*1*) *CptnModNestEnv*(*7*)
unfolding *env-tran-right-def Sta-def* **by** *fastforce*
ultimately show *?thesis*
using *CptnModNestEnv*(*3*) *CptnModNestEnv*(*7*) *CptnModNestEnv*(*8*) *CptnModNestEnv*(*9*) *env-tran-tail* **by** *blast*
qed
then obtain *xs* **where** *hi*:(Γ, *xs*) $\in$ *cpn n* Γ *P t1* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*
by *fastforce*
have *s1-s*:*s1*=*s* **using** *CptnModNestEnv* **unfolding** *cpn-def* **by** *auto*
obtain *xsa′* **where** *xs*:*xs*=((*P*,*t1*)#*xsa′*) $\wedge$ (*n*, Γ,((*P*,*t1*)#*xsa′*))$\in$*cptn-mod-nest-call* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*t1*)#*xsa′*)
using *hi* **unfolding** *cpn-def* **by** *fastforce*
have *env-tran*:$\Gamma\vdash_c$(*P*,*s1*) $\rightarrow_e$ (*P*,*t1*) **using** *CptnModNestEnv Seq-env-P* **by** (*metis fst-conv nth-Cons-0*)
then have (*n*, Γ,(*P*,*s1*)#(*P*,*t1*)#*xsa′*)$\in$*cptn-mod-nest-call*
using *xs env-tran cptn-mod-nest-call.CptnModNestEnv* **by** *fastforce*
then have (Γ,(*P*,*s1*)#(*P*,*t1*)#*xsa′*) $\in$ *cpn n* Γ *P s*
using *cpn-def s1-s* **by** *fastforce*
moreover have (*C*,*s1*)#(*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*s1*)#(*P*,*t1*)#*xsa′*)
using *xs C* **unfolding** *lift-def* **by** *fastforce*
ultimately show *?case* **by** *auto*
next
case (*CptnModNestSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestThrow*)

thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestSeq1 n* Γ *P0 sa xsa zs P1*)
then have *a1*:*LanguageCon.com.Seq P Q* = *LanguageCon.com.Seq P0 P1*
by *fastforce*
have *f1*: *sa* = *s*
using *CptnModNestSeq1.prems*(*1*) **by** *force*
have *f2*: *P* = *P0* ∧ *Q* = *P1* **using** *a1* **by** *auto*
hence (Γ, (*P0*, *sa*) # *xsa*) ∈ *cpn n* Γ *P s*
using *f2 f1 CptnModNestSeq1.hyps*(*1*) **by** (*simp add*: *cpn-def*)
thus *?case*
using *Cons-lift CptnModNestSeq1.hyps*(*3*) *a1* **by** *fastforce*
next
case (*CptnModNestSeq2 n* Γ *P0 sa xsa P1 ys zs*)
then have *P0* = *P* ∧ *P1* = *Q* **by** *auto*
then obtain *i* **where** *zs*:*fst* (*zs*!*i*) = *Q* ∧ (*i*< (*length zs*)) **using** *CptnModNestSeq2*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ fst-conv length-Cons length-append nth-append-length zero-less-Suc zero-less-diff*)
then have *Suc i*< *length* ((*Seq P0 P1*,*sa*)#*zs*) **by** *fastforce*
then have *fst* (((*Seq P0 P1*, *sa*) # *zs*)!*Suc i*) = *Q* **using** *zs* **by** *fastforce*
thus *?case* **using** *CptnModNestSeq2*(*8*) *zs* **by** *auto*
next
case (*CptnModNestSeq3 n* Γ *P1 sa xsa s′ ys zs Q1*)
have *s′-a*:*s′* ∈ *a*
proof −
have *cpP1*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cpn n* Γ *P1* (*Normal sa*)
using *CptnModNestSeq3.hyps*(*1*) **unfolding** *cpn-def* **by** *fastforce*
then have *cpP1′*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cp* Γ *P1* (*Normal sa*)
using *CptnModNestSeq3.hyps*(*1*) *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod*

unfolding *cp-def* **by** *fastforce*
have *map*:((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*) = *map* (*lift Q1*) ((*P1*, *Normal sa*) # *xsa*)
using *CptnModSeq3* **by** (*simp add*: *Cons-lift*)
then
have (Γ,((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*))) ∈ *assum* (*p*,*rely*)
proof −
have *env-tran-right* Γ ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*)) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *map*
by (*metis* (*no-types*) *Cons-lift-append CptnModNestSeq3.hyps*(*7*) *CptnModNestSeq3.prems*(*4*) *env-tran-subr*)
thus *?thesis* **using** *CptnModNestSeq3*(*12*)
unfolding *assum-def env-tran-right-def* **by** *fastforce*
qed
moreover have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cpn n* Γ (*Seq P1 Q1*) (*Normal sa*)
using *CptnModNestSeq3*(*7*) *CptnModNestSeq3.hyps*(*1*) *cptn-mod-nest-call.CptnModNestSeq1*

unfolding *cpn-def* **by** *fastforce*
then have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cp* Γ (*Seq P1 Q1*) (*Normal sa*)
using *CptnModNestSeq3.hyps*(*1*) *cptn-eq-cptn-mod-set cptn-mod.CptnModSeq1 cptn-mod-nest-cptn-mod*
unfolding *cp-def* **by** *fastforce*
ultimately have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *assum* (*p*,*rely*)
using *assum-map map cpP1′* **by** *fastforce*
then have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *comm* (*G*,(*q*,*a*)) *F*
using *cpP1 CptnModNestSeq3*(*13*) *CptnModNestSeq3.prems*(*1*) **unfolding** *com-validityn-def* **by** *auto*
thus *?thesis*
using *CptnModNestSeq3*(*3*) *CptnModNestSeq3*(*4*)
unfolding *comm-def final-def* **by** *fastforce*
qed
have *final* (*last* ((*LanguageCon.com.Throw*, *Normal s′*)# *ys*))
proof −
have *cptn-mod*:(*n*, Γ, (*LanguageCon.com.Throw*, *Normal s′*) # *ys*) ∈ *cptn-mod-nest-call*

using *CptnModNestSeq3*(*5*) **by** (*simp add*: *cptn-eq-cptn-mod-set*)
then have *cptn*:(Γ, (*LanguageCon.com.Throw*, *Normal s′*) # *ys*) ∈ *cptn*
using *cptn-eq-cptn-mod-nest* **by** *auto*
moreover have *throw-0*:((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)!*0* = (*Throw*, *Normal s′*) ∧ *0* < *length*((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)
by *force*
moreover have *last*:*last* ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*) = ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)!((*length* ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)) − *1*)
using *last-conv-nth* **by** *auto*
moreover have *env-tran*:*env-tran-right* Γ ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *env-tran-subl env-tran-tail* **by** *blast*
ultimately obtain *st′* **where** *fst* (*last* ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)) = *Throw* ∧
snd (*last* ((*LanguageCon.com.Throw*, *Normal s′*) # *ys*)) = *Normal st′*
using *zero-throw-all-throw*[*of* Γ ((*Throw*, *Normal s′*) # *ys*) *s′* (*length* ((*Throw*, *Normal s′*) # *ys*))−*1 a rely*]
s′-a CptnModNestSeq3(*11*) *CptnModNestSeq3*(*12*) **by** *fastforce*
thus *?thesis* **using** *CptnModNestSeq3*(*10*) *final-def* **by** *blast*
qed
thus *?case* **using** *CptnModNestSeq3*(*10*) *CptnModNestSeq3*(*7*)
by *force*
qed (*auto*)

lemma *Seq-sound2*:
assumes
a0:(Γ,*x*)∈*cptn-mod* **and**

a1:*x*!*0* = ((*Seq P Q*),*s*) **and**
a2:$\forall$ *i*<*length x*. *fst* (*x*!*i*)$\neq$ *Q* **and**
a3:*fst* (*last x*) = *Throw* $\wedge$ *snd* (*last x*) = *Normal s'* **and**
a4:*env-tran-right* Γ *x rely*
shows
$\exists$ *xs s' ys*. (Γ,*xs*) $\in$ *cp* Γ *P s* $\wedge$ *x* = ((*map* (*lift Q*) *xs*)@((*Throw*, *Normal s'*)#*ys*))
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *P s s'*)
case (*CptnModOne* Γ *C s1*)
then have (Γ, [(*P*,*s*)]) $\in$ *cp* Γ *P s* $\wedge$ [(*C*, *s1*)] = *map* (*lift Q*) [(*P*,*s*)]@[(*Throw*, *Normal s'*)]
unfolding *cp-def lift-def* **by** (*simp add*: *cptn.CptnOne*)
thus *?case* **by** *fastforce*
next
case (*CptnModEnv* Γ *C s1 t1 xsa*)
then have *C*:*C*=*Seq P Q* **unfolding** *lift-def* **by** *fastforce*
have $\exists$ *xs s' ys*. (Γ, *xs*) $\in$ *cp* Γ *P t1* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*@((*Throw*, *Normal s'*)#*ys*)
proof −
have ((*C*, *t1*) # *xsa*) ! *0* = (*LanguageCon.com.Seq P Q*, *t1*) **using** *C* **by** *auto*
moreover have $\forall$ *i*<*length* ((*C*, *t1*) # *xsa*). *fst* (((*C*, *t1*) # *xsa*) ! *i*) $\neq$ *Q*
using *CptnModEnv*(*5*) **by** *fastforce*
moreover have *fst* (*last* ((*C*, *t1*) # *xsa*)) = *Throw* $\wedge$ *snd* (*last* ((*C*, *t1*) # *xsa*)) = *Normal s'* **using** *CptnModEnv*(*6*)
by *fastforce*
ultimately show *?thesis*
using *CptnModEnv*(*3*) *CptnModEnv*(*7*) *env-tran-tail* **by** *blast*
qed
then obtain *xs s'' ys* **where** *hi*:(Γ, *xs*) $\in$ *cp* Γ *P t1* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*@((*Throw*, *Normal s''*)#*ys*)
by *fastforce*
have *s1-s*:*s1*=*s* **using** *CptnModEnv* **unfolding** *cp-def* **by** *auto*
have $\exists$ *xsa' s'' ys*. *xs*=((*P*,*t1*)#*xsa'*) $\wedge$ (Γ,((*P*,*t1*)#*xsa'*))$\in$*cptn* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*t1*)#*xsa'*)@((*Throw*, *Normal s''*)#*ys*)
using *hi* **unfolding** *cp-def*
proof −
have (Γ,*xs*)$\in$*cptn* $\wedge$ *xs*!*0* = (*P*,*t1*) **using** *hi* **unfolding** *cp-def* **by** *fastforce*
moreover then have *xs*$\neq$[] **using** *cptn.simps* **by** *fastforce*
ultimately obtain *xsa'* **where** *xs*=((*P*,*t1*)#*xsa'*) **using** *SmallStepCon.nth-tl* **by** *fastforce*
thus *?thesis*
using *hi* **using** ‹(Γ, *xs*) $\in$ *cptn* $\wedge$ *xs* ! *0* = (*P*, *t1*)› **by** *auto*
qed
then obtain *xsa' s'' ys* **where** *xs*:*xs*=((*P*,*t1*)#*xsa'*) $\wedge$ (Γ,((*P*,*t1*)#*xsa'*))$\in$*cptn* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*t1*)#*xsa'*)@((*Throw*, *Normal s''*)#*ys*)
by *fastforce*
have *env-tran*:$\Gamma\vdash_c$(*P*,*s1*) $\rightarrow_e$ (*P*,*t1*) **using** *CptnModEnv Seq-env-P* **by** (*metis fst-conv nth-Cons-0*)

then have $(\Gamma,(P,s1)\#(P,t1)\#xsa')\in cptn$ **using** *xs env-tran CptnEnv* **by** *fast-force*
then have $(\Gamma,(P,s1)\#(P,t1)\#xsa') \in cp\ \Gamma\ P\ s$
using *cp-def s1-s* **by** *fastforce*
moreover have $(C,s1)\#(C,\ t1)\ \#\ xsa = map\ (lift\ Q)\ ((P,s1)\#(P,t1)\#xsa')@((Throw,\ Normal\ s'')\#ys)$
using *xs C* **unfolding** *lift-def* **by** *fastforce*
ultimately show *?case* **by** *auto*
next
case (*CptnModSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModThrow*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModSeq1* Γ *P0 sa xsa zs P1*)
thus *?case*
proof −
have $a1{:}\forall\ c\ p.\ fst\ (case\ p\ of\ (ca{::}('s,\ 'a,\ 'd,'e)\ LanguageCon.com,\ x{::}('s,\ 'd)\ xstate) \Rightarrow$
$(LanguageCon.com.Seq\ ca\ c,\ x)) = LanguageCon.com.Seq\ (fst\ p)\ c$
by *simp*
then have $[] = xsa$
proof −
have $[] \neq zs$
using *CptnModSeq1* **by** *force*
then show *?thesis*
by (*metis* (*no-types*) *LanguageCon.com.distinct*(*71*) *One-nat-def CptnModSeq1*(*3*,*6*)
last.simps last-conv-nth last-lift)
qed
then have $\forall\ c.\ Throw = c \lor [] = zs$
using *CptnModSeq1*(*3*) **by** *fastforce*
then show *?thesis*
using *CptnModSeq1.prems*(*3*) **by** *force*
qed
next
case (*CptnModSeq2* Γ *P0 sa xsa P1 ys zs*)
then have $P0 = P \land P1 = Q$ **by** *auto*
then obtain *i* **where** $zs{:}fst\ (zs!i) = Q \land (i< (length\ zs))$ **using** *CptnModSeq2*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ fst-conv length-Cons length-append nth-append-length zero-less-Suc zero-less-diff*)
then have $Suc\ i< length\ ((Seq\ P0\ P1,sa)\#zs)$ **by** *fastforce*
then have $fst\ (((Seq\ P0\ P1,\ sa)\ \#\ zs)!Suc\ i) = Q$ **using** *zs* **by** *fastforce*
thus *?case* **using** *CptnModSeq2*(*8*) *zs* **by** *auto*
next
case (*CptnModSeq3* Γ *P0 sa xsa s″ ys zs P1*)
then have $P0 = P \land P1 = Q \land s{=}Normal\ sa$ **by** *auto*
moreover then have $(\Gamma,\ (P0,\ Normal\ sa)\ \#\ xsa){\in}\ cp\ \Gamma\ P\ s$

using *CptnModSeq3*(*1*)
by (*simp add*: *cp-def cptn-eq-cptn-mod-set*)
moreover have *last zs*=(*Throw*, *Normal s'*) **using** *CptnModSeq3*(*10*) *CptnModSeq3.hyps*(*7*)
by (*simp add*: *prod-eqI*)
ultimately show *?case* **using** *CptnModSeq3*(*7*)
using *Cons-lift-append* **by** *blast*
qed (*auto*)

lemma *Seq-sound2'*:
assumes
a0:(*n*,Γ,*x*)∈*cptn-mod-nest-call* **and**
a1:*x*!*0* = ((*Seq P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:*fst* (*last x*) = *Throw* ∧ *snd* (*last x*) = *Normal s'* **and**
a4:*env-tran-right* Γ *x rely*
shows
∃ *xs s' ys*. (Γ,*xs*) ∈ *cpn n* Γ *P s* ∧ *x* = ((*map* (*lift Q*) *xs*)@((*Throw*, *Normal s'*)#*ys*))
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *P s s'*)
case (*CptnModNestOne n* Γ *C s1*)
then have (Γ, [(*P*,*s*)]) ∈ *cpn n* Γ *P s* ∧ [(*C*, *s1*)] = *map* (*lift Q*) [(*P*,*s*)]@[(*Throw*, *Normal s'*)]
unfolding *cp-def lift-def* **by** (*simp add*: *cptn.CptnOne*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
then have *C*:*C*=*Seq P Q* **unfolding** *lift-def* **by** *fastforce*
have ∃ *xs s' ys*. (Γ, *xs*) ∈ *cpn n* Γ *P t1* ∧ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*@((*Throw*, *Normal s'*)#*ys*)
proof −
have ((*C*, *t1*) # *xsa*) ! *0* = (*LanguageCon.com.Seq P Q*, *t1*) **using** *C* **by** *auto*
moreover have ∀ *i*<*length* ((*C*, *t1*) # *xsa*). *fst* (((*C*, *t1*) # *xsa*) ! *i*) ≠ *Q*
using *CptnModNestEnv*(*5*) **by** *fastforce*
moreover have *fst* (*last* ((*C*, *t1*) # *xsa*)) = *Throw* ∧ *snd* (*last* ((*C*, *t1*) # *xsa*)) = *Normal s'*
using *CptnModNestEnv*(*6*)
by *fastforce*
ultimately show *?thesis*
using *CptnModNestEnv*(*3*) *CptnModNestEnv*(*7*) *env-tran-tail* **by** *blast*
qed
then obtain *xs s'' ys* **where** *hi*:(Γ, *xs*) ∈ *cpn n* Γ *P t1* ∧ (*C*, *t1*) # *xsa* = *map* (*lift Q*) *xs*@((*Throw*, *Normal s''*)#*ys*)
by *fastforce*
have *s1-s*:*s1*=*s* **using** *CptnModNestEnv* **unfolding** *cp-def* **by** *auto*
have ∃ *xsa' s'' ys*. *xs*=((*P*,*t1*)#*xsa'*) ∧ (*n*, Γ,((*P*,*t1*)#*xsa'*))∈*cptn-mod-nest-call*

$\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*t1*)#*xsa'*)@((*Throw*, *Normal s''*)#*ys*)
using *hi* **unfolding** *cp-def*
proof −
have (*n*, Γ,*xs*)∈*cptn-mod-nest-call* ∧ *xs*!*0* = (*P*,*t1*) **using** *hi* **unfolding** *cpn-def* **by** *fastforce*
moreover then have *xs*≠[] **using** *cptn-mod-nest-call.simps* **by** *fastforce*
ultimately obtain *xsa'* **where** *xs*=((*P*,*t1*)#*xsa'*) **using** *SmallStepCon.nth-tl* **by** *fastforce*
thus *?thesis*
using *hi* **using** ‹(*n*, Γ, *xs*) ∈ *cptn-mod-nest-call* ∧ *xs* ! *0* = (*P*, *t1*)› **by** *auto*

qed
then obtain *xsa' s'' ys* **where** *xs*:*xs*=((*P*,*t1*)#*xsa'*) ∧ (*n*, Γ,((*P*,*t1*)#*xsa'*))∈*cptn-mod-nest-call* ∧
(*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*t1*)#*xsa'*)@((*Throw*, *Normal s''*)#*ys*)
by *fastforce*
have *env-tran*:$\Gamma\vdash_c$(*P*,*s1*) $\rightarrow_e$ (*P*,*t1*) **using** *CptnModNestEnv Seq-env-P* **by** (*metis fst-conv nth-Cons-0*)
then have (*n*, Γ,(*P*,*s1*)#(*P*,*t1*)#*xsa'*)∈*cptn-mod-nest-call* **using** *xs env-tran cptn-mod-nest-call.CptnModNestEnv* **by** *blast*
then have (Γ,(*P*,*s1*)#(*P*,*t1*)#*xsa'*) ∈ *cpn n* Γ *P s*
using *cpn-def s1-s* **by** *fastforce*
moreover have (*C*,*s1*)#(*C*, *t1*) # *xsa* = *map* (*lift Q*) ((*P*,*s1*)#(*P*,*t1*)#*xsa'*)@((*Throw*, *Normal s''*)#*ys*)
using *xs C* **unfolding** *lift-def* **by** *fastforce*
ultimately show *?case* **by** *auto*
next
case (*CptnModNestSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestThrow*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestSeq1 n* Γ *P0 sa xsa zs P1*)
thus *?case*
proof −
have *a1*:∀ *c p*. *fst* (*case p of* (*ca*::('*s*, '*a*, '*d*,'*e*) *LanguageCon.com*, *x*::('*s*, '*d*) *xstate*) ⇒
(*LanguageCon.com.Seq ca c*, *x*)) = *LanguageCon.com.Seq* (*fst p*) *c*
by *simp*
then have [] = *xsa*
proof −
have [] ≠ *zs*
using *CptnModNestSeq1* **by** *force*
then show *?thesis*
by (*metis* (*no-types*) *LanguageCon.com.distinct*(*71*) *One-nat-def CptnModNestSeq1*(*3*,*6*)
last.simps last-conv-nth last-lift)

qed
then have ∀ *c*. *Throw* = *c* ∨ [] = *zs*
using *CptnModNestSeq1*(*3*) **by** *fastforce*
then show *?thesis*
using *CptnModNestSeq1.prems*(*3*) **by** *force*
qed
next
case (*CptnModNestSeq2 n* Γ *P0 sa xsa P1 ys zs*)
then have *P0* = *P* ∧ *P1* = *Q* **by** *auto*
then obtain *i* **where** *zs*:*fst* (*zs*!*i*) = *Q* ∧ (*i*< (*length zs*)) **using** *CptnModNestSeq2*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ fst-conv length-Cons length-append nth-append-length zero-less-Suc zero-less-diff*)
then have *Suc i*< *length* ((*Seq P0 P1*,*sa*)#*zs*) **by** *fastforce*
then have *fst* (((*Seq P0 P1*, *sa*) # *zs*)!*Suc i*) = *Q* **using** *zs* **by** *fastforce*
thus *?case* **using** *CptnModNestSeq2*(*8*) *zs* **by** *auto*
next
case (*CptnModNestSeq3 n* Γ *P0 sa xsa s′′ ys zs P1*)
then have *P0* = *P* ∧ *P1* = *Q* ∧ *s*=*Normal sa* **by** *auto*
moreover then have (Γ, (*P0*, *Normal sa*) # *xsa*)∈ *cpn n* Γ *P s*
using *CptnModNestSeq3*(*1*)
by (*simp add*: *cpn-def*)
moreover have *last zs*=(*Throw*, *Normal s′*) **using** *CptnModNestSeq3*(*10*) *CptnModNestSeq3.hyps*(*7*)
by (*simp add*: *prod-eqI*)
ultimately show *?case* **using** *CptnModNestSeq3*(*7*)
using *Cons-lift-append* **by** *blast*
qed (*auto*)

lemma *Last-Skip-Exist-Final*:
assumes
a0:(Γ,*x*)∈*cptn* **and**
a1:*x*!*0* = ((*Seq P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:*fst*(*last x*) = *Skip*
shows
∃ *c s′ i*. *i*<*length x* ∧ *x*!*i* = (*Seq c Q*,*s′*) ∧ *final* (*c*,*s′*)
using *a0 a1 a2 a3*
proof (*induct arbitrary*: *P s*)
case (*CptnOne* Γ *c s1*) **thus** *?case* **by** *fastforce*
next
case (*CptnEnv* Γ *C st t xsa*)
thus *?case*
proof −
have *LanguageCon.com.Seq P Q* = *C*
using *CptnEnv.prems*(*1*) **by** *auto*
then show *?thesis*
using *CptnEnv.hyps*(*3*) *CptnEnv.prems*(*2*) *CptnEnv.prems*(*3*) **by** *fastforce*
qed
next

case (*CptnComp* Γ *C st C′ st′ xsa*)
then have *c-seq*:*C* = (*Seq P Q*) ∧ *st* = *s* **by** *force*
from *CptnComp* **show** *?case* **proof**(*cases*)
case (*Seqc P1 P1′ P2*)
then have ∃ *c s′ i*. *i* < *length* ((*C′*, *st′*) # *xsa*) ∧
((*C′*, *st′*) # *xsa*) ! *i* = (*LanguageCon.com.Seq c Q*, *s′*) ∧
SmallStepCon.final (*c*, *s′*)
using *CptnComp last.simps* **by** *fastforce*
thus *?thesis* **by** *fastforce*
next
case (*SeqThrowc C2 s′*)
thus *?thesis*
proof −
have *LanguageCon.com.Seq LanguageCon.com.Throw Q* = *C*
using ⟨*C* = *LanguageCon.com.Seq LanguageCon.com.Throw C2*⟩ *c-seq* **by** *blast*
then show *?thesis*
using ⟨*st* = *Normal s′*⟩ **unfolding** *final-def* **by** *force*
qed
next
case (*FaultPropc*) **thus** *?thesis*
using *c-seq redex-not-Seq* **by** *blast*
next
case (*StuckPropc*) **thus** *?thesis*
using *c-seq redex-not-Seq* **by** *blast*
next
case (*AbruptPropc*) **thus** *?thesis*
using *c-seq redex-not-Seq* **by** *blast*
qed (*auto*)
qed

lemma *Seq-sound3*:
assumes
a0:(*n*,Γ,*x*)∈*cptn-mod-nest-call* **and**
a1:*x*!*0* = ((*Seq P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:*fst*(*last x*) = *Skip* **and**
a4:*env-tran-right* Γ *x rely* **and**
a5:*snd* (*x*!*0*)∈ *Normal* ‘ *p* ∧ *Sta p rely* ∧ *Sta a rely* **and**
a6: Γ $\models n_{/F}$ *P sat* [*p*, *rely*, *G*, *q*,*a*]
shows
False
using *a0 a1 a2 a3 a4 a5 a6*
proof (*induct arbitrary*: *P s p*)
case (*CptnModNestOne n* Γ *C s1*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
then have *C*:*C*=*Seq P Q* **unfolding** *lift-def* **by** *fastforce*

thus *?case*
proof −
have $((C, t1) \# xsa) ! 0 = (LanguageCon.com.Seq\ P\ Q, t1)$ **using** *C* **by** *auto*
moreover have $\forall i<length\ ((C, t1) \# xsa).\ fst\ (((C, t1) \# xsa) ! i) \neq Q$
using *CptnModNestEnv(5)* **by** *fastforce*
moreover have $fst\ (last\ ((C, t1) \# xsa)) = LanguageCon.com.Skip$ **using** *CptnModNestEnv(6)*
by (*simp add*: *SmallStepCon.final-def*)
moreover have $snd\ (((C, t1) \# xsa) ! 0) \in Normal\ `\ p$
using *CptnModNestEnv(8) CptnModNestEnv(1) CptnModNestEnv(7)*
unfolding *env-tran-right-def Sta-def* **by** *fastforce*
ultimately show *?thesis*
using *CptnModNestEnv(3) CptnModNestEnv(7) CptnModNestEnv(8) CptnModNestEnv(9) env-tran-tail*
by *blast*
qed
next
case (*CptnModNestSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestThrow*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestSeq1 n* Γ *P0 sa xsa zs P1*)
obtain *cl* **where** $fst\ (last\ ((LanguageCon.com.Seq\ P0\ P1, sa) \# zs)) = Seq\ cl\ P1$
using *CptnModNestSeq1(3)* **by** (*metis One-nat-def fst-conv last.simps last-conv-nth last-lift map-is-Nil-conv*)
thus *?case* **using** *CptnModNestSeq1(6)* **by** *auto*
next
case (*CptnModNestSeq2 n* Γ *P0 sa xsa P1 ys zs*)
then have $P0 = P \wedge P1 = Q$ **by** *auto*
then obtain *i* **where** $zs{:}fst\ (zs!i) = Q \wedge (i< (length\ zs))$ **using** *CptnModNestSeq2*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ fst-conv length-Cons length-append nth-append-length zero-less-Suc zero-less-diff*)
thus *?case* **using** *CptnModNestSeq2(8) zs* **by** *auto*
next
case (*CptnModNestSeq3 n* Γ *P1 sa xsa s′ ys zs Q1*)
have $s'\text{-}a{:}s' \in\ \ a$
proof −
have *cpnP1*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cpn n* Γ *P1* (*Normal sa*)
using *CptnModNestSeq3.hyps(1)* **unfolding** *cpn-def*
by *fastforce*
then have *cpP1*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cp* Γ *P1* (*Normal sa*)
using *CptnModNestSeq3.hyps(1) cptn-mod-nest-cptn-mod cptn-if-cptn-mod* **unfolding** *cp-def cpn-def*
by *fastforce*
have *map*:((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*) = *map* (*lift Q1*)

((*P1*, *Normal sa*) # *xsa*)
using *CptnModNestSeq3* **by** (*simp add*: *Cons-lift*)
then
have (Γ,((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*))) ∈ *assum* (*p*,*rely*)
proof −
have *env-tran-right* Γ ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*)) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *map*
by (*metis* (*no-types*) *Cons-lift-append CptnModNestSeq3.hyps*(*7*) *CptnModNestSeq3.prems*(*4*) *env-tran-subr*)
thus *?thesis* **using** *CptnModNestSeq3*(*12*)
unfolding *assum-def env-tran-right-def* **by** *fastforce*
qed
moreover have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cpn n* Γ (*Seq P1 Q1*) (*Normal sa*)
using *CptnModNestSeq3.hyps*(*1*)
CptnModNestSeq1
unfolding *cpn-def* **by** *fastforce*
then have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cp* Γ (*Seq P1 Q1*) (*Normal sa*)
using *CptnModNestSeq3.hyps*(*1*) *cp-def cptn-eq-cptn-mod-set*
cptn-mod.CptnModSeq1 cptn-mod-nest-cptn-mod **by** *fastforce*
ultimately have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *assum* (*p*,*rely*)
using *assum-map map cpP1* **by** *fastforce*
then have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *comm* (*G*,(*q*,*a*)) *F*
using *cpnP1 CptnModNestSeq3*(*13*) *CptnModNestSeq3.prems*(*1*) **unfolding** *com-validityn-def* **by** *auto*
thus *?thesis*
using *CptnModNestSeq3*(*3*) *CptnModNestSeq3*(*4*)
unfolding *comm-def final-def* **by** *fastforce*
qed
have *fst* (*last* ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)) = *Throw*
proof −
have *cptn*:(Γ, (*LanguageCon.com.Throw*, *Normal s'*) # *ys*) ∈ *cptn*
using *CptnModNestSeq3*(*5*)
using *cptn-eq-cptn-mod-nest* **by** *blast*
moreover have *throw-0*:((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)!*0* = (*Throw*, *Normal s'*) ∧ *0* < *length*((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)
by *force*
moreover have *last*:*last* ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*) = ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)!((*length* ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)) − *1*)
using *last-conv-nth* **by** *auto*
moreover have *env-tran*:*env-tran-right* Γ ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *env-tran-subl env-tran-tail* **by** *blast*
ultimately obtain *st'* **where** *fst* (*last* ((*LanguageCon.com.Throw*, *Normal s'*)

ys)) = *Throw* ∧
snd (last ((LanguageCon.com.Throw, Normal s') # ys)) = Normal st'
using *zero-throw-all-throw*[of Γ ((Throw, Normal s') # ys) s' (length ((Throw, Normal s') # ys))−1 a rely]
s'-a CptnModNestSeq3(11) CptnModNestSeq3(12) **by** *fastforce*
thus *?thesis* **using** *CptnModNestSeq3(10) final-def* **by** *blast*
qed
thus *?case* **using** *CptnModNestSeq3(10) CptnModNestSeq3(7)*
by *force*
qed(*auto*)

lemma *map-xs-ys*:
assumes
a0:(Γ, (P0, sa) # xsa) ∈ cptn-mod **and**
a1:fst (last ((P0, sa) # xsa)) = C **and**
a2:(Γ, (P1, snd (last ((P0, sa) # xsa))) # ys) ∈ cptn-mod **and**
a3:zs = map (lift P1) xsa @ (P1, snd (last ((P0, sa) # xsa))) # ys **and**
a4:((LanguageCon.com.Seq P0 P1, sa) # zs) ! 0 = (LanguageCon.com.Seq P Q, s) **and**
a5:i < length ((LanguageCon.com.Seq P0 P1, sa) # zs) ∧ ((LanguageCon.com.Seq P0 P1, sa) # zs) ! i = (Q, sj) **and**
a6:∀ j<i. fst (((LanguageCon.com.Seq P0 P1, sa) # zs) ! j) ≠ Q
shows
∃ xs ys. (Γ, xs) ∈ cp Γ P s ∧
(Γ, ys) ∈ cp Γ Q (snd (xs ! (i − 1))) ∧ (LanguageCon.com.Seq P0 P1, sa) # zs = map (lift Q) xs @ ys
proof −
let ?P0 = (P0, sa) # xsa
have P-Q:P=P0 ∧ s=sa ∧ Q = P1 **using** a4 **by** *force*
have i:i=(length ((P0, sa) # xsa))
proof (*cases* i=(length ((P0, sa) # xsa)))
case *True* **thus** *?thesis* **by** *auto*
next
case *False*
then have i:i<(length ((P0, sa) # xsa)) ∨ i > (length ((P0, sa) # xsa)) **by** *auto*
{
assume i:i<(length ((P0, sa) # xsa))
then have eq-map:((LanguageCon.com.Seq P0 P1, sa) # zs) ! i = map (lift P1) ((P0, sa) # xsa) ! i
using *a3 Cons-lift-append* **by** (*metis* (*no-types*, *lifting*) *length-map nth-append*)

then have ∃ ci si. map (lift P1) ((P0, sa) # xsa) ! i = (Seq ci P1,si)
using *i* **unfolding** *lift-def*
proof −
have map (λ(c, y). (LanguageCon.com.Seq c P1, y)) ((P0, sa) # xsa) ! i = (case ((P0, sa) # xsa) ! i of (c, x) ⇒ (LanguageCon.com.Seq c P1, x))
by (*meson* ‹i < length ((P0, sa) # xsa)› *nth-map*)

then show $\exists c\ x.\ map\ (\lambda(c, x).\ (LanguageCon.com.Seq\ c\ P1,\ x))\ ((P0,\ sa)\ \#\ xsa)\ !\ i = (LanguageCon.com.Seq\ c\ P1,\ x)$
by (*simp add*: *case-prod-beta*)
qed
then have $((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs)\ !\ i \neq (Q,\ sj)$
using *P-Q eq-map* **by** *fastforce*
then have *?thesis* **using** *a5* **by** *auto*
}note *l=this*
{
assume *i*:$i>(length\ ((P0,\ sa)\ \#\ xsa))$
have $fst\ (((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs)\ !\ (length\ ?P0)) = Q$
using *a3 P-Q Cons-lift-append* **by** (*metis fstI length-map nth-append-length*)

then have *?thesis* **using** *a6 i* **by** *auto*
}
thus *?thesis* **using** *l i* **by** *auto*
qed
then have $(\Gamma,\ (P0,\ sa)\ \#\ xsa) \in cp\ \Gamma\ P\ s$
using *a0 cptn-eq-cptn-mod P-Q* **unfolding** *cp-def* **by** *fastforce*
also have $(\Gamma,\ (P1,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in cp\ \Gamma\ Q\ (snd\ (?P0\ !\ ((length\ ?P0)\ -1)))$
using *a3 cptn-eq-cptn-mod P-Q* **unfolding** *cp-def*
proof −
have $(\Gamma,\ (Q,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in cptn\text{-}mod$
using *a2 P-Q* **by** *blast*
then have $(\Gamma,\ (Q,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in \{(f,\ ps).\ ps\ !\ 0 = (Q,\ snd\ (((P0,\ sa)\ \#\ xsa)\ !\ (Suc\ (length\ xsa)\ -\ 1))) \wedge (\Gamma,\ ps) \in cptn \wedge f = \Gamma\}$
by (*simp add*: *cptn-eq-cptn-mod last-length*)
then show $(\Gamma,\ (P1,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in \{(f,\ ps).\ ps\ !\ 0 = (Q,\ snd\ (((P0,\ sa)\ \#\ xsa)\ !\ (length\ ((P0,\ sa)\ \#\ xsa)\ -\ 1))) \wedge (\Gamma,\ ps) \in cptn \wedge f = \Gamma\}$
using *P-Q* **by** *force*
qed
ultimately show *?thesis* **using** *a3 P-Q i* **using** *Cons-lift-append* **by** *blast*
qed

lemma *map-xs-ys'*:
assumes
a0:$(n,\ \Gamma,\ (P0,\ sa)\ \#\ xsa) \in cptn\text{-}mod\text{-}nest\text{-}call$ **and**
a1:$fst\ (last\ ((P0,\ sa)\ \#\ xsa)) = C$ **and**
a2:$(n,\Gamma,\ (P1,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in cptn\text{-}mod\text{-}nest\text{-}call$ **and**
a3:$zs = map\ (lift\ P1)\ xsa\ @\ (P1,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys$ **and**
a4:$((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs)\ !\ 0 = (LanguageCon.com.Seq\ P\ Q,\ s)$ **and**
a5:$i < length\ ((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs) \wedge ((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs)\ !\ i = (Q,\ sj)$ **and**
a6:$\forall j<i.\ fst\ (((LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs)\ !\ j) \neq Q$
shows
$\exists xs\ ys.\ (\Gamma,\ xs) \in cpn\ n\ \Gamma\ P\ s\ \wedge$

(Γ, *ys*) ∈ *cpn n* Γ *Q* (*snd* (*xs* ! (*i* − *1*))) ∧ (*LanguageCon.com.Seq P0 P1*, *sa*) # *zs* = *map* (*lift Q*) *xs* @ *ys*
proof −
let *?P0* = (*P0*, *sa*) # *xsa*
have *P-Q*:*P*=*P0* ∧ *s*=*sa* ∧ *Q* = *P1* **using** *a4* **by** *force*
have *i*:*i*=(*length* ((*P0*, *sa*) # *xsa*))
proof (*cases i*=(*length* ((*P0*, *sa*) # *xsa*)))
case *True* **thus** *?thesis* **by** *auto*
next
case *False*
then have *i*:*i*<(*length* ((*P0*, *sa*) # *xsa*)) ∨ *i* > (*length* ((*P0*, *sa*) # *xsa*)) **by** *auto*
{
assume *i*:*i*<(*length* ((*P0*, *sa*) # *xsa*))
then have *eq-map*:((*LanguageCon.com.Seq P0 P1*, *sa*) # *zs*) ! *i* = *map* (*lift P1*) ((*P0*, *sa*) # *xsa*) ! *i*
using *a3 Cons-lift-append* **by** (*metis* (*no-types*, *lifting*) *length-map nth-append*)

then have ∃ *ci si*. *map* (*lift P1*) ((*P0*, *sa*) # *xsa*) ! *i* = (*Seq ci P1*,*si*)
using *i* **unfolding** *lift-def*
proof −
have *map* (λ(*c*, *y*). (*LanguageCon.com.Seq c P1*, *y*)) ((*P0*, *sa*) # *xsa*) ! *i* = (*case* ((*P0*, *sa*) # *xsa*) ! *i of* (*c*, *x*) ⇒ (*LanguageCon.com.Seq c P1*, *x*))
by (*meson* ‹*i* < *length* ((*P0*, *sa*) # *xsa*)› *nth-map*)
then show ∃ *c x*. *map* (λ(*c*, *x*). (*LanguageCon.com.Seq c P1*, *x*)) ((*P0*, *sa*) # *xsa*) ! *i* = (*LanguageCon.com.Seq c P1*, *x*)
by (*simp add*: *case-prod-beta*)
qed
then have ((*LanguageCon.com.Seq P0 P1*, *sa*) # *zs*) ! *i* ≠ (*Q*, *sj*)
using *P-Q eq-map* **by** *fastforce*
then have *?thesis* **using** *a5* **by** *auto*
}note *l*=*this*
{
assume *i*:*i*>(*length* ((*P0*, *sa*) # *xsa*))
have *fst* (((*LanguageCon.com.Seq P0 P1*, *sa*) # *zs*) ! (*length ?P0*)) = *Q*
using *a3 P-Q Cons-lift-append* **by** (*metis fstI length-map nth-append-length*)

then have *?thesis* **using** *a6 i* **by** *auto*
}
thus *?thesis* **using** *l i* **by** *auto*
qed
then have (Γ, (*P0*, *sa*) # *xsa*) ∈ *cpn n* Γ *P s*
using *a0 P-Q* **unfolding** *cpn-def* **by** *fastforce*
also have (Γ, (*P1*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ *cpn n* Γ *Q* (*snd* (*?P0* ! ((*length ?P0*) −*1*)))
using *a3 cptn-eq-cptn-mod P-Q* **unfolding** *cpn-def*
proof −
have (*n*, Γ, (*Q*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call*
using *a2 P-Q* **by** *blast*

then have $(\Gamma,\ (Q,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in \{(f,\ ps).\ ps\ !\ 0 = (Q,\ snd\ (((P0,\ sa)\ \#\ xsa)\ !\ (Suc\ (length\ xsa)\ -\ 1)))\ \wedge$
$(n,\ \Gamma,\ ps) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge f = \Gamma\}$
by (*simp add*: *cptn-eq-cptn-mod last-length*)
then show $(\Gamma,\ (P1,\ snd\ (last\ ((P0,\ sa)\ \#\ xsa)))\ \#\ ys) \in \{(f,\ ps).\ ps\ !\ 0 = (Q,\ snd\ (((P0,\ sa)\ \#\ xsa)\ !\ (length\ ((P0,\ sa)\ \#\ xsa)\ -\ 1)))\ \wedge (n,\Gamma,\ ps) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge f = \Gamma\}$
using *P-Q* **by** *force*
qed
ultimately show *?thesis* **using** *a3 P-Q i* **using** *Cons-lift-append* **by** *blast*
qed

lemma *Seq-sound4*:
assumes
a0:$(n,\Gamma,x)\in cptn\text{-}mod\text{-}nest\text{-}call$ **and**
a1:$x!0 = ((Seq\ P\ Q),s)$ **and**
a2:$i<length\ x \wedge x!i=(Q,sj)$ **and**
a3:$\forall j<i.\ fst(x!j)\neq Q$ **and**
a4:*env-tran-right* Γ *x rely* **and**
a5:$snd\ (x!0)\in$ *Normal ‘ p* $\wedge$ *Sta p rely* $\wedge$ *Sta a rely* **and**
a6: $\Gamma \models n_{/F}\ P\ sat\ [p,\ rely,\ G,\ q,a]$
shows
$\exists xs\ ys.\ (\Gamma,xs) \in (cpn\ n\ \Gamma\ P\ s) \wedge (\Gamma,ys) \in (cpn\ n\ \Gamma\ Q\ (snd\ (xs!(i-1)))) \wedge x = (map\ (lift\ Q)\ xs)@ys$
using *a0 a1 a2 a3 a4 a5 a6*
proof (*induct arbitrary*: *i sj P s p*)
case (*CptnModNestOne* Γ *C s1*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C st t n xsa*)
have *a1*:$Seq\ P\ Q \neq Q$ **by** *simp*
then have *C-seq*:$C=(Seq\ P\ Q)$ **using** *CptnModNestEnv* **by** *fastforce*
then have $fst(((C,\ st)\ \#\ (C,\ t)\ \#\ xsa)!0) \neq Q$ **using** *CptnEnv a1* **by** *auto*
moreover have *n-q*:$fst(((C,\ st)\ \#\ (C,\ t)\ \#\ xsa)!1) \neq Q$ **using** *CptnModNestEnv a1* **by** *auto*
moreover have $fst(((C,\ st)\ \#\ (C,\ t)\ \#\ xsa)!i) = Q$ **using** *CptnModNestEnv* **by** *auto*
ultimately have *i-suc*: $i> (Suc\ 0)$
by (*metis Suc-eq-plus1 Suc-lessI add.left-neutral neq0-conv*)
then obtain i' **where** i':$i=Suc\ i'$ **by** (*meson lessE*)
then have *i-minus*:$i'=i-1$ **by** *auto*
have *c-init*:$((C,\ t)\ \#\ xsa)\ !\ 0 = ((Seq\ P\ Q),\ t)$
using *CptnModNestEnv* **by** *auto*
moreover have $i'< length\ ((C,t)\#xsa) \wedge ((C,t)\#xsa)!i' = (Q,sj)$
using i' *CptnModNestEnv*(5) **by** *force*
moreover have $\forall j<i'.\ fst\ (((C,\ t)\ \#\ xsa)\ !\ j) \neq Q$
using i' *CptnModNestEnv*(6) **by** *force*

moreover have *snd* $(((C,\ t)\ \#\ xsa)\ !\ 0) \in Normal$ ' p
using *CptnModNestEnv*(8) *CptnModNestEnv*(1) *CptnModNestEnv*(7)
unfolding *env-tran-right-def Sta-def* **by** *fastforce*
ultimately have *hyp*:$\exists xs\ ys.$
$(\Gamma,\ xs) \in cpn\ n\ \Gamma\ P\ t \wedge$
$(\Gamma,\ ys) \in cpn\ n\ \Gamma\ Q\ (snd\ (xs\ !\ (i'-1))) \wedge (C,\ t)\ \#\ xsa = map\ (lift\ Q)\ xs\ @$
ys
using *CptnModNestEnv*(3) *env-tran-tail* *CptnModNestEnv*(8) *CptnModNestEnv*(9)

CptnModNestEnv.prems(4) **by** *blast*
then obtain *xs ys* **where** *xs-cp*:$(\Gamma,\ xs) \in cpn\ n\ \Gamma\ P\ t \wedge$
$(\Gamma,\ ys) \in cpn\ n\ \Gamma\ Q\ (snd\ (xs\ !\ (i'-1))) \wedge (C,\ t)\ \#\ xsa = map\ (lift\ Q)\ xs\ @$
ys
by *fast*
have $(\Gamma,\ (P,s)\#xs) \in cpn\ n\ \Gamma\ P\ s$
proof −
have $xs!0 = (P,t)$
using *xs-cp* **unfolding** *cpn-def* **by** *blast*
moreover have $xs\neq[]$
using *xs-cp n-q c-init* **unfolding** *cpn-def* **by** *auto*
ultimately obtain xs' **where** xs':$(n,\ \Gamma,\ (P,t)\#xs') \in cptn\text{-}mod\text{-}nest\text{-}call \wedge$
$xs=(P,t)\#xs'$
using *SmallStepCon.nth-tl xs-cp* **unfolding** *cpn-def* **by** *force*
thus *?thesis*
proof −
have $(LanguageCon.com.Seq\ P\ Q,\ s) = (C,\ st)$
using *CptnModNestEnv.prems*(1) **by** *auto*
then have $\Gamma\vdash_c (P,\ s) \rightarrow_e (P,\ t)$
using *Seq-env-P CptnModNestEnv*(1) **by** *blast*
then show *?thesis*
by (*simp add*:xs' *cpn-def cptn-mod-nest-call.CptnModNestEnv*)
qed
qed
thus *?case*
using *i-suc Cons-lift-append CptnModNestEnv.prems*(1) i' *i-minus xs-cp*
by *fastforce*
next
case (*CptnModNestSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestThrow*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Seq fst-conv nth-Cons-0*)
next
case (*CptnModNestSeq1* $n\ \Gamma$ *P0 sa xsa zs P1*)
then have *P1-Q*:$P1 = Q$ **by** *auto*
let $?x = (LanguageCon.com.Seq\ P0\ P1,\ sa)\ \#\ zs$
have $\forall j<length\ ?x.\ \exists c\ s.\ ?x!j = (Seq\ c\ P1,s)$ **using** *CptnModNestSeq1*(3)
proof (*induct xsa arbitrary*: *zs P0 P1 sa*)
case *Nil* **thus** *?case* **by** *auto*

next
case (*Cons a xsa*)
then obtain *ac as* **where** *a*=(*ac*,*as*) **by** *fastforce*
then have *zs*:*zs* = (*Seq ac P1*,*as*)#(*map* (*lift P1*) *xsa*)
using *Cons*(*2*)
unfolding *lift-def* **by** *auto*
have *zs-eq*:(*map* (*lift P1*) *xsa*)=(*map* (*lift P1*) *xsa*) **by** *auto*
note *hyp*=*Cons*(*1*)[*OF zs-eq*]
note *hyp*[*of ac as*]
thus *?case* **using** *zs Cons*(*2*) **by** (*metis One-nat-def diff-Suc-Suc diff-zero length-Cons less-Suc-eq-0-disj nth-Cons'*)
qed
thus *?case* **using** *P1-Q CptnModNestSeq1*(*5*) **using** *fstI seq-not-eq2* **by** *auto*
next
case (*CptnModNestSeq2 n* Γ *P0 sa xsa P1 ys zs*)
then show *?case* **using** *map-xs-ys'*[*OF CptnModNestSeq2*(*1*) *CptnModNestSeq2*(*3*) *CptnModNestSeq2*(*4*) *CptnModNestSeq2*(*6*)
CptnModNestSeq2(*7*) *CptnModNestSeq2*(*8*) *CptnModNestSeq2*(*9*)] **by** *blast*
next
case (*CptnModNestSeq3 n* Γ *P1 sa xsa s' ys zs Q1*)
then have *P-Q*:*P*=*P1* ∧ *Q* = *Q1* **by** *force*
thus *?case*
proof (*cases Q1* = *Throw*)
case *True* **thus** *?thesis* **using** *map-xs-ys'*[*of n* Γ *P1 Normal sa xsa Throw Throw ys zs*]
CptnModNestSeq3 **by** *fastforce*
next
case *False* **note** *q-not-throw*=*this*
have ∀ *x*. *x*< *length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ⟶
((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x* ≠ (*Q*, *sj*)
proof −
{
fix *x*
assume *x-less*:*x*< *length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*)
have ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x* ≠ (*Q*, *sj*)
proof (*cases x* < *length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*)#*map* (*lift Q1*) *xsa*))
case *True*
then have *eq-map*:((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x* = *map* (*lift Q1*) ((*P1*, *Normal sa*) # *xsa*) ! *x*
by (*metis* (*no-types*) *Cons-lift Cons-lift-append CptnModNestSeq3*.*hyps*(*7*) *True nth-append*)
then have ∃ *ci si*. *map* (*lift Q1*) ((*P1*, *Normal sa*) # *xsa*) ! *x* = (*Seq ci Q1*,*si*)
using *True* **unfolding** *lift-def*
proof −
have *x* < *length* ((*P1*, *Normal sa*) # *xsa*)
using *True* **by** *auto*

then have *map* (λ(*c*, *y*). (*LanguageCon.com.Seq c Q1*, *y*)) ((*P1*, *Normal sa*) # *xsa*) ! *x* = (*case* ((*P1*, *Normal sa*) # *xsa*) ! *x of* (*c*, *x*) ⇒ (*LanguageCon.com.Seq c Q1*, *x*))
using *nth-map* **by** *blast*
then show ∃ *c x1*. *map* (λ(*c*, *x1*). (*LanguageCon.com.Seq c Q1*, *x1*)) ((*P1*, *Normal sa*) # *xsa*) ! *x* = (*LanguageCon.com.Seq c Q1*, *x1*)
by (*simp add*: *case-prod-beta′*)
qed
then have ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x* ≠ (*Q*, *sj*)
using *P-Q eq-map* **by** *fastforce*
thus *?thesis* **using** *CptnModNestSeq3*(*10*) **by** *auto*
next
case *False*
have *s′-a*:*s′* ∈ *a*
proof −
have *cpP1*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cpn n* Γ *P1* (*Normal sa*)
using *CptnModNestSeq3*.*hyps*(*1*) *cptn-eq-cptn-mod-set* **unfolding** *cpn-def* **by** *fastforce*
then have *cpP1′*:(Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *cp* Γ *P1* (*Normal sa*)
unfolding *cpn-def cp-def*
using *cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *fastforce*
have *map*:((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*) = *map* (*lift Q1*) ((*P1*, *Normal sa*) # *xsa*)
using *CptnModSeq3* **by** (*simp add*: *Cons-lift*)
then
have (Γ,((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*))) ∈ *assum* (*p*,*rely*)
proof −
have *env-tran-right* Γ ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # (*map* (*lift Q1*) *xsa*)) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *map*
by (*metis* (*no-types*) *Cons-lift-append CptnModNestSeq3*.*hyps*(*7*) *CptnModNestSeq3*.*prems*(*4*) *env-tran-subr*)
thus *?thesis* **using** *CptnModNestSeq3*(*12*)
unfolding *assum-def env-tran-right-def* **by** *fastforce*
qed
moreover have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cpn n* Γ (*Seq P1 Q1*) (*Normal sa*)
using *CptnModNestSeq3*(*7*) *CptnModNestSeq3*.*hyps*(*1*) *cptn-eq-cptn-mod-set cptn-mod-nest-call*.*CptnModNestSeq1*
unfolding *cpn-def* **by** *fastforce*
then have (Γ,((*Seq P1 Q1*), *Normal sa*)#(*map* (*lift Q1*) *xsa*)) ∈ *cp* Γ (*Seq P1 Q1*) (*Normal sa*)
unfolding *cpn-def cp-def*
by (*simp add*: *cptn-if-cptn-mod cptn-mod-nest-cptn-mod*)
ultimately have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *assum* (*p*,*rely*)
using *assum-map map cpP1′* **by** *fastforce*
then have (Γ, (*P1*, *Normal sa*) # *xsa*) ∈ *comm* (*G*,(*q*,*a*)) *F*

using *cpP1 CptnModNestSeq3*(*13*) *CptnModNestSeq3.prems*(*1*) **unfolding** *com-validityn-def* **by** *auto*
thus *?thesis*
using *CptnModNestSeq3*(*3*) *CptnModNestSeq3*(*4*)
unfolding *comm-def final-def* **by** *fastforce*
qed
have *all-throw*:$\forall$ *i*<*length* ((*LanguageCon.com.Throw*, *Normal s'*)# *ys*).
fst (((*LanguageCon.com.Throw*, *Normal s'*)# *ys*)!*i*) = *Throw*
proof −
{**fix** *i*
assume *i*:*i*< *length* ((*LanguageCon.com.Throw*, *Normal s'*)# *ys*)
have *cptn*:(*n*, Γ, (*LanguageCon.com.Throw*, *Normal s'*) # *ys*) ∈ *cptn-mod-nest-call*

using *CptnModNestSeq3*(*5*) **by** *auto*
moreover have *throw-0*:((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)!*0* = (*Throw*, *Normal s'*) ∧ *0* < *length*((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)
by *force*
moreover have *last*:*last* ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*) = ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)!((*length* ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*)) − *1*)
using *last-conv-nth* **by** *auto*
moreover have *env-tran*:*env-tran-right* Γ ((*LanguageCon.com.Throw*, *Normal s'*) # *ys*) *rely*
using *CptnModNestSeq3*(*11*) *CptnModNestSeq3*(*7*) *env-tran-subl env-tran-tail* **by** *blast*
ultimately have
fst (((*LanguageCon.com.Throw*, *Normal s'*)# *ys*)!*i*) = *Throw*
using *zero-throw-all-throw*[*of* Γ ((*Throw*, *Normal s'*) # *ys*) *s' i a rely*]
s'-a *CptnModNestSeq3*(*12*) *i*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *auto*
}
thus *?thesis* **using** *CptnModNestSeq3*(*10*) *final-def* **by** *blast*
qed
then have
$\forall$ *x*≥ *length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *map* (*lift Q1*) *xsa*).
x<*length* (((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*)) ⟶
fst (((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x*) = *Throw*
proof−
{
fix *x*
assume *a1*:*x*≥ *length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *map* (*lift Q1*) *xsa*) **and**
a2:*x*<*length* (((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*))
then have ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x* =
((*LanguageCon.com.Throw*, *Normal s'*)# *ys*) !(*x* − (*length* ((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *map* (*lift Q1*) *xsa*)))
using *CptnModNestSeq3*(*7*) **by** (*metis Cons-lift Cons-lift-append not-le nth-append*)

then have*fst* (((*LanguageCon.com.Seq P1 Q1*, *Normal sa*) # *zs*) ! *x*) = *Throw*
 using *all-throw a1 a2 CptnModNestSeq3.hyps*(*7*) **by** *auto*
 } **thus** *?thesis* **by** *auto*
 qed
 thus *?thesis* **using** *False CptnModNestSeq3*(*7*) *q-not-throw P-Q x-less*
 by (*metis fst-conv not-le*)
 qed
 } **thus** *?thesis* **by** *auto*
 qed
 thus *?thesis* **using** *CptnModNestSeq3*(*9*) **by** *fastforce*
 qed
qed(*auto*)

inductive-cases *stepc-elim-cases-Seq-throw*:
$\Gamma\vdash_c(Seq\ c1\ c2,s) \rightarrow (Throw,\ Normal\ s1)$

inductive-cases *stepc-elim-cases-Seq-skip-c2*:
$\Gamma\vdash_c(Seq\ c1\ c2,s) \rightarrow (c2,s)$

lemma *seq-skip-throw*:
$\Gamma\vdash_c(Seq\ c1\ c2,s) \rightarrow (c2,s) \implies c1= Skip \vee (c1=Throw \wedge (\exists s2'.\ s=Normal\ s2'))$
apply (*rule stepc-elim-cases-Seq-skip-c2*)
apply *fastforce*
apply (*auto*)+
apply (*fastforce intro*:*redex-not-Seq*)+
done

lemma *Seq-sound*:
$\Gamma,\Theta \vdash_{/F} c1\ sat\ [p,\ R,\ G,\ q,a] \implies$
$\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p,\ R,\ G,\ q,a] \implies$
$\Gamma,\Theta \vdash_{/F} c2\ sat\ [q,\ R,\ G,\ r,a] \implies$
$\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [q,\ R,\ G,\ r,a] \implies$
$Sta\ a\ R \wedge Sta\ p\ R \implies (\forall s.\ (Normal\ s,Normal\ s) \in G) \implies$
$\Gamma,\Theta \models n_{/F}\ (Seq\ c1\ c2)\ sat\ [p,\ R,\ G,\ r,a]$
proof −
 assume
 a0:$\Gamma,\Theta \vdash_{/F} c1\ sat\ [p,\ R,\ G,\ q,a]$ **and**
 a1:$\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p,\ R,\ G,\ q,a]$ **and**
 a2:$\Gamma,\Theta \vdash_{/F} c2\ sat\ [q,\ R,\ G,\ r,a]$ **and**
 a3: $\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [q,\ R,\ G,\ r,a]$ **and**
 a4: $Sta\ a\ R \wedge Sta\ p\ R$ **and**
 a5: $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$
 {
 fix *s*

assume *all-call*:$\forall (c,p,R,G,q,a) \in \Theta.\ \Gamma \models n_{/F}\ (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]$
then have *a1*:$\Gamma \models n_{/F}\ c1\ sat\ [p,\ R,\ G,\ q,a]$
using *a1 com-cvalidityn-def* **by** *fastforce*
then have *a3*: $\Gamma \models n_{/F}\ c2\ sat\ [q,\ R,\ G,\ r,a]$
using *a3 com-cvalidityn-def all-call* **by** *fastforce*
have $cpn\ n\ \Gamma\ (Seq\ c1\ c2)\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (r,a))\ F$
proof −
{
fix *c*
assume *a10*:$c \in cpn\ n\ \Gamma\ (Seq\ c1\ c2)\ s$ **and** *a11*:$c \in assum(p,\ R)$
then have *a10'*:$c \in cp\ \Gamma\ (Seq\ c1\ c2)\ s$ **unfolding** *cpn-def cp-def*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
obtain $\Gamma 1\ l$ **where** *c-prod*:$c=(\Gamma 1,l)$ **by** *fastforce*
have *cp*:$l!0=((Seq\ c1\ c2),s) \wedge (\Gamma,l) \in cptn \wedge \Gamma=\Gamma 1$ **using** *a10' cp-def c-prod* **by** *fastforce*
have *cptn-nest*:$l!0=((Seq\ c1\ c2),s) \wedge (n,\Gamma,l) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge \Gamma=\Gamma 1$ **using** *a10 cpn-def c-prod* **by** *fastforce*
have $\Gamma 1$:$(\Gamma,\ l) = c$ **using** *c-prod cp* **by** *blast*
have $c \in comm(G,\ (r,a))\ F$
proof −
{
assume *l-f*:$snd\ (last\ l) \notin Fault\ `\ F$
have *assum*:$snd(l!0) \in Normal\ `\ (p) \wedge (\forall i.\ Suc\ i<length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:$env\text{-}tran\ \Gamma\ p\ l\ R$ **using** *env-tran-def cp* **by** *blast*
then have *env-tran-right*: $env\text{-}tran\text{-}right\ \Gamma\ l\ R$
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have $(\forall i.\ Suc\ i<length\ l \longrightarrow$
$\Gamma\vdash_c(l!i)\ \ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)\wedge$
$(final\ (last\ l)\ \ \longrightarrow$
$((fst\ (last\ l) = Skip\ \wedge$
$snd\ (last\ l) \in Normal\ `\ \ r)) \vee$
$(fst\ (last\ l) = Throw\ \wedge$
$snd\ (last\ l) \in Normal\ `\ a))$
proof $(cases\ \forall i<length\ l.\ fst\ (l!i) \neq c2)$
case *True*
then have *no-c2*:$\forall i<length\ l.\ fst\ (l!i) \neq c2$ **by** *assumption*
show *?thesis*
proof $(cases\ final\ (last\ l))$
case *True*
then obtain s' **where** $fst\ (last\ l) = Skip \vee (fst\ (last\ l) = Throw \wedge snd\ (last\ l) = Normal\ s')$
using *final-def* **by** *fast*
thus *?thesis*
proof
assume $fst\ (last\ l) = LanguageCon.com.Skip$

then have *False*
using *no-c2 env-tran-right cptn-nest cptn-eq-cptn-mod-set Seq-sound3 a4 a1 assum*
by *blast*
thus *?thesis* **by** *auto*
next
assume *asm0*:*fst* (*last l*) = *LanguageCon.com.Throw* ∧ *snd* (*last l*) = *Normal s′*
then obtain *lc1 s1′ ys* **where** *cpn-lc1*:(Γ,*lc1*) ∈ *cpn n* Γ *c1 s* ∧ *l* = ((*map* (*lift c2*) *lc1*)@((*Throw*, *Normal s1′*)#*ys*))
using *Seq-sound2′*[*of n* Γ *l c1 c2 s s′*] *cptn-nest cptn-eq-cptn-mod-set env-tran-right no-c2* **by** *blast*
then have *cp-lc1*:(Γ,*lc1*) ∈ *cp* Γ *c1 s*
using *cptn-if-cptn-mod cptn-mod-nest-cptn-mod split-conv*
unfolding *cp-def cpn-def* **by** *blast*
let *?m-lc1* = *map* (*lift c2*) *lc1*
let *?lm-lc1* = (*length ?m-lc1*)
let *?last-m-lc1* = *?m-lc1*!(*?lm-lc1*−*1*)
have *lc1-not-empty*:*lc1* ≠ []
using Γ*1 a10 cpn-lc1 cp* **by** *auto*
then have *map-cpn*:(Γ,*?m-lc1*) ∈ *cpn n* Γ (*Seq c1 c2*) *s*
proof −
have *f1*: *lc1* ! *0* = (*c1*, *s*) ∧ (*n*,Γ, *lc1*) ∈ *cptn-mod-nest-call* ∧ Γ = Γ
using *cpn-lc1 cpn-def* **by** *blast*
then have *f2*: (*n*, Γ, *?m-lc1*) ∈ *cptn-mod-nest-call*
by (*metis* (*no-types*) *Cons-lift cptn-mod-nest-call.CptnModNestSeq1 f1 lc1-not-empty list.exhaust nth-Cons-0*)
then show *?thesis*
using *f2 f1 lc1-not-empty* **by** (*simp add*: *cpn-def lift-def*)
qed
then have *map-cp*:(Γ,*?m-lc1*) ∈ *cp* Γ (*Seq c1 c2*) *s*
by (*metis* (*no-types*, *lifting*) *cp-def cp-lc1 cpn-def lift-is-cptn mem-Collect-eq split-conv*)
also have *map-assum*:(Γ,*?m-lc1*) ∈ *assum* (*p*,*R*)
using *sub-assum a10 a11* Γ*1 cpn-lc1 lc1-not-empty*
by (*metis SmallStepCon.nth-tl map-is-Nil-conv*)
ultimately have ((Γ,*lc1*) ∈ *assum*(*p*, *R*))
using Γ*1 assum-map cp-lc1* **by** *blast*
then have *lc1-comm*:(Γ,*lc1*) ∈ *comm*(*G*, (*q*,*a*)) *F*
using *a1 cpn-lc1* **unfolding** *com-validityn-def* **by** *blast*
then have *m-lc1-comm*:(Γ,*?m-lc1*) ∈ *comm*(*G*, (*q*,*a*)) *F*
using *map-cp map-assum comm-map cp-lc1* **by** *fastforce*
then have *last-m-lc1*:*last* (*?m-lc1*) = (*Seq* (*fst* (*last lc1*)) *c2*,*snd* (*last lc1*))
proof −
have *a000*:∀ *p c*. (*LanguageCon.com.Seq* (*fst p*) *c*, *snd p*) = *lift c p*
using *Cons-lift* **by** *force*
then show *?thesis*
by (*simp add*: *last-map a000 lc1-not-empty*)

qed
then have *last-length*:*last* (*?m-lc1*) = *?last-m-lc1*
using *lc1-not-empty last-conv-nth list.map-disc-iff* **by** *blast*
then have *l-map*:*l*!(*?lm-lc1*−*1*)= *?last-m-lc1*
using *cpn-lc1*
by (*simp add*:*lc1-not-empty nth-append*)
then have *lm-lc1*:*l*!(*?lm-lc1*) = (*Throw*, *Normal s1′*)
using *cpn-lc1* **by** (*meson nth-append-length*)
then have *step*:$\Gamma\vdash_c$(*l*!(*?lm-lc1*−*1*)) → (*l*!(*?lm-lc1*))
proof −
have $\Gamma\vdash_c$(*l*!(*?lm-lc1*−*1*)) $\rightarrow_{ce}$ (*l*!(*?lm-lc1*))
proof −
have *f1*: ∀ *n na*. ¬ *n* < *na* ∨ *Suc* (*na* − *Suc n*) = *na* − *n*
by (*meson Suc-diff-Suc*)
have *map* (*lift c2*) *lc1* ≠ []
by (*metis lc1-not-empty map-is-Nil-conv*)
then have *f2*: *0* < *length* (*map* (*lift c2*) *lc1*)
by (*meson length-greater-0-conv*)
then have *length* (*map* (*lift c2*) *lc1*) − *1* + *1* < *length* (*map* (*lift c2*) *lc1* @ (*LanguageCon.com.Throw*, *Normal s1′*) # *ys*)
by *simp*
then show *?thesis*
using *f2 f1*
by (*metis Suc-pred′ cp cpn-lc1 cptn-tran-ce-i*)
qed
moreover have ¬$\Gamma\vdash_c$(*l*!(*?lm-lc1*−*1*)) $\rightarrow_e$ (*l*!(*?lm-lc1*))
using *last-m-lc1 last-length l-map*
proof −
have (*LanguageCon.com.Seq* (*fst* (*last lc1*)) *c2*, *snd* (*last lc1*)) = *l* ! (*length* (*map* (*lift c2*) *lc1*) − *1*)
using *l-map last-m-lc1 local.last-length* **by** *presburger*
then show *?thesis*
by (*metis* (*no-types*) *LanguageCon.com.distinct*(*71*) ‹*l* ! *length* (*map* (*lift c2*) *lc1*) = (*LanguageCon.com.Throw*, *Normal s1′*)› *env-c-c′*)
qed
ultimately show *?thesis* **using** *step-ce-elim-cases* **by** *blast*
qed
then have *last-lc1-suc*:*snd* (*l*!(*?lm-lc1*−*1*)) = *snd* (*l*!*?lm-lc1*) ∧ *fst* (*l*!(*?lm-lc1*−*1*)) = *Seq Throw c2*
using *lm-lc1 stepc-elim-cases-Seq-throw*
by (*metis One-nat-def asm0 append-is-Nil-conv cpn-lc1 diff-Suc-less fst-conv l-map last-conv-nth last-m-lc1 length-greater-0-conv list.simps*(*3*) *local.last-length no-c2 snd-conv*)
then have *a-normal*:*snd* (*l*!*?lm-lc1*) ∈ *Normal* ‘ (*a*)
proof
have *last-lc1*:*fst* (*last lc1*) = *Throw* ∧ *snd* (*last lc1*) = *Normal s1′*
using *last-length l-map lm-lc1 last-m-lc1 last-lc1-suc*
by (*metis LanguageCon.com.inject*(*3*) *fst-conv snd-conv*)
have *final* (*last lc1*) **using** *last-lc1 final-def*

```
    by blast
  moreover have snd (last lc1)∉ Fault ‘ F
    using last-lc1 by fastforce
  ultimately have (fst (last lc1) = Throw ∧
        snd (last lc1) ∈ Normal ‘ (a))
    using lc1-comm last-lc1 unfolding comm-def by force
  thus ?thesis using  l-map last-lc1-suc last-m-lc1 last-length by auto
qed
have concl:(∀ i. Suc i<length l ⟶
  Γ⊢c(l!i)  → (l!(Suc i)) ⟶
    (snd(l!i), snd(l!(Suc i))) ∈ G)
proof−
{ fix k ns ns'
  assume a00:Suc k<length l and
   a21:Γ⊢c(l!k)  → (l!(Suc k))
   then have i-m-l:∀ i <?lm-lc1  . l!i = ?m-lc1!i
     using cp-lc1
   proof −
     have map (lift c2) lc1 ≠ []
       by (meson lc1-not-empty list.map-disc-iff)
     then show ?thesis
       by (metis (no-types) cpn-lc1  nth-append)
   qed
   have last-not-F:snd (last ?m-lc1) ∉ Fault ‘ F
      using l-map last-lc1-suc lm-lc1 last-length by auto
   have (snd(l!k), snd(l!(Suc k))) ∈ G
   proof (cases Suc k< ?lm-lc1)
     case True
     then have a11': Γ⊢c(?m-lc1!k)  → (?m-lc1!(Suc k))
       using a11 i-m-l True
     proof −
       have ∀ n na. ¬ 0 < n − Suc na ∨ na < n
         using diff-Suc-eq-diff-pred zero-less-diff by presburger
       then show ?thesis
         by (metis (no-types)  True a21 i-m-l  zero-less-diff)
     qed
     then have (snd(?m-lc1!k), snd(?m-lc1!(Suc k))) ∈ G
    using a11' m-lc1-comm True comm-dest1 l-f last-not-F by fastforce
     thus ?thesis using i-m-l using  True by fastforce
   next
     case False
     then have (Suc k=?lm-lc1) ∨ (Suc k>?lm-lc1) by auto
     thus ?thesis
     proof
       {assume suck:(Suc k=?lm-lc1)
        then have k:k=?lm-lc1−1 by auto
        have G-s1':(Normal s1', Normal s1')∈G
          using a5 by auto
        then show (snd (l!k), snd (l!Suc k)) ∈ G
```

proof –
have *snd* (*l*!*Suc k*) = *Normal s1'*
using *lm-lc1 suck* **by** *fastforce*
then show *?thesis* **using** *suck k G-s1' last-lc1-suc* **by** *fastforce*
qed
}
next
{
assume *a001*:*Suc k*>*?lm-lc1*
have $\forall$ *i*. *i*$\geq$(*length lc1*) $\wedge$ (*Suc i* < *length l*) $\longrightarrow$
$\neg$($\Gamma\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*)))
using *lm-lc1 lc1-not-empty*
proof –
have *env-tran-right* Γ *l R*
by (*metis env-tran-right*)
then show *?thesis*
using *a-normal cp fst-conv length-map*
lm-lc1 only-one-component-tran-j[*of* Γ *l ?lm-lc1 s1' a k R*]
snd-conv a21 a001 a00
a4 **by** *auto*
qed
then have $\neg$($\Gamma\vdash_c$(*l*!*k*) $\rightarrow$ (*l*!(*Suc k*)))
using *a00 a001* **by** *auto*
then show *?thesis* **using** *a21* **by** *fastforce*
}
qed
qed
} **thus** *?thesis* **by** *auto*
qed
have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *r*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *a*))
proof –
have *l-t*:*fst* (*last l*) = *Throw*
using *lm-lc1* **by** (*simp add*: *asm0*)
have *?lm-lc1* $\leq$ *length l* –*1* **using** *cpn-lc1* **by** *fastforce*
then have *snd* (*l* ! (*length l* – *1*)) $\in$ *Normal* ‘ *a*
using *cp a-normal a4 fst-conv lm-lc1 snd-conv*
env-tran-right i-throw-all-throw[*of* Γ *l ?lm-lc1 s1'* (*length l* –*1*)
- *R a*]
by (*metis* (*no-types*, *lifting*) *One-nat-def diff-is-0-eq diff-less diff-less-Suc diff-zero image-iff length-greater-0-conv lessI less-antisym list*.*size*(*3*) *xstate*.*inject*(*1*))
thus *?thesis* **using** *l-t*
by (*simp add*: *cpn-lc1 last-conv-nth*)
qed
note *res* = *conjI* [*OF concl concr*]

then show *?thesis* **using** $\Gamma 1$ *c-prod* **unfolding** *comm-def* **by** *auto*
qed
next
case *False*
then obtain *lc1* **where** *cpn-lc1*:$(\Gamma, lc1) \in cpn\ n\ \Gamma\ c1\ s \wedge l = map\ (lift\ c2)\ lc1$
using *Seq-sound1 assum False no-c2 env-tran-right cptn-nest cptn-eq-cptn-mod-set a4 a1*
by *blast*
then have *cp-lc1*:$(\Gamma, lc1) \in cp\ \Gamma\ c1\ s$
using *cp-def cpn-def cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *fastforce*
then have $((\Gamma, lc1) \in assum(p,\ R))$
using $\Gamma 1$ *cpn-lc1 a10' a11 assum-map* **by** *blast*
then have $(\Gamma,\ lc1) \in comm(G,\ (q,a))\ F$ **using** *cpn-lc1 a1*
by (*meson IntI com-validityn-def contra-subsetD*)
then have $(\Gamma,\ l) \in comm(G,\ (r,a))\ F$
using *comm-map a10'* $\Gamma 1$ *cp-lc1 cpn-lc1* **by** *blast*
then show *?thesis* **using** *l-f*
unfolding *comm-def* **by** *auto*
qed
next
case *False*
then obtain *k* **where** *k-len*:$k < length\ l \wedge fst\ (l\ !\ k) = c2$
by *blast*
then have $\exists m.\ (m < length\ l \wedge fst\ (l\ !\ m) = c2) \wedge$
$(\forall i<m.\ \neg\ (i < length\ l \wedge fst\ (l\ !\ i) = c2))$
using *a0 exists-first-occ*$[of\ (\lambda i.\ i<length\ l\ \wedge fst\ (l\ !\ i) = c2)\ k]$
by *blast*
then obtain *i* **where** *a0*:$i<length\ l \wedge fst\ (l\ !i) = c2 \wedge$
$(\forall j<i.\ (fst\ (l\ !\ j) \neq c2))$
by *fastforce*
then obtain *s2* **where** *li*:$l!i = (c2,s2)$ **by** (*meson eq-fst-iff*)
then obtain *lc1 lc2* **where** *cp-lc1*:$(\Gamma, lc1) \in (cpn\ n\ \Gamma\ c1\ s) \wedge$
$(\Gamma, lc2) \in (cpn\ n\ \Gamma\ c2\ (snd\ (lc1!(i-1)))) \wedge$
$l = (map\ (lift\ c2)\ lc1)@lc2$
using *Seq-sound4*$[of\ n\ \Gamma\ l\ c1\ c2\ s]$ *a0 env-tran-right a4 a1 cptn-nest assum*
by *blast*
then have *cp-lc1'*:$(\Gamma, lc1) \in (cp\ \ \Gamma\ c1\ s) \wedge$
$(\Gamma, lc2) \in (cp\ \ \Gamma\ c2\ (snd\ (lc1!(i-1))))$
unfolding *cp-def cpn-def cptn-eq-cptn-mod-nest* **by** *fastforce*
have $\forall i < length\ l.\ snd\ (l!i) \notin Fault\ `\ F$
using *cp l-f last-not-F*$[of\ \Gamma\ l\ F]$ **by** *blast*
then have *i-not-fault*:$snd\ (l!i) \notin Fault\ `\ F$ **using** *a0* **by** *blast*
have *length-c1-map*:$length\ lc1 = length\ (map\ (lift\ c2)\ lc1)$
by *fastforce*
then have *i-map*:$i=length\ lc1$
using *cp-lc1 li a0* **unfolding** *lift-def*
proof −

assume *a1*: (Γ, *lc1*) ∈ *cpn n* Γ *c1 s* ∧ (Γ, *lc2*) ∈ *cpn n* Γ *c2* (*snd* (*lc1* ! (*i* − *1*))) ∧ *l* = *map* (λ(*P*, *s*). (*LanguageCon.com.Seq P c2*, *s*)) *lc1* @ *lc2*
have *f2*: *i* < *length l* ∧ *fst* (*l* ! *i*) = *c2* ∧ (∀ *n*. ¬ *n* < *i* ∨ *fst* (*l* ! *n*) ≠ *c2*)
using *a0* **by** *blast*
have *f3*: (*LanguageCon.com.Seq* (*fst* (*lc1* ! *i*)) *c2*, *snd* (*lc1* ! *i*)) = *lift c2* (*lc1* ! *i*)
by (*simp add*: *case-prod-unfold lift-def*)
then have *fst* (*l* ! *length lc1*) = *c2*
using *a1* **by** (*simp add*: *cpn-def nth-append*)
thus *?thesis*
using *f3 f2* **by** (*metis* (*no-types*) *nth-append cp-lc1 fst-conv length-map lift-nth linorder-neqE-nat seq-and-if-not-eq*(*4*))
qed
have *lc2-l*:∀ *j*<*length lc2*. *lc2*!*j*=*l*!(*i*+*j*)
using *cp-lc1 length-c1-map i-map a0*
by (*metis nth-append-length-plus*)
have *lc1-not-empty*:*lc1* ≠ []
using *cp cp-lc1* **unfolding** *cpn-def* **by** *fastforce*
have *lc2-not-empty*:*lc2* ≠ []
using *a0 cp-lc1 i-map* **by** *auto*
have *l-is*:*s2*= *snd* (*last lc1*)
using *cp-lc1 li a0 lc1-not-empty i-map* **unfolding** *cpn-def*
by (*auto simp add*: *last-conv-nth lc2-l*)
let *?m-lc1* = *map* (*lift c2*) *lc1*

have *last-m-lc1*:*l*!(*i*−*1*) = (*Seq* (*fst* (*last lc1*)) *c2*,*s2*)
proof −
have *a000*:∀ *p c*. (*LanguageCon.com.Seq* (*fst p*) *c*, *snd p*) = *lift c p*
using *Cons-lift* **by** *force*
have *length* (*map* (*lift c2*) *lc1*) = *i*
using *i-map* **by** *fastforce*
then show *?thesis*
by (*metis* (*no-types*) *One-nat-def l-is a000 cp-lc1 diff-less last-conv-nth last-map lc1-not-empty length-c1-map length-greater-0-conv less-Suc0 nth-append*)

qed
have *last-mcl1-not-F*:*snd* (*last ?m-lc1*) ∉ *Fault* ‘ *F*
proof −
have *map* (*lift c2*) *lc1* ≠ []
by (*metis lc1-not-empty list.map-disc-iff*)
then show *?thesis*
by (*metis* (*full-types*) *One-nat-def i-not-fault l-is last-conv-nth last-snd lc1-not-empty li snd-conv*)
qed
have *map-cp*:(Γ,*?m-lc1*) ∈ *cpn n* Γ (*Seq c1 c2*) *s*
proof −
have *f1*: *lc1* ! *0* = (*c1*, *s*) ∧ (*n*,Γ, *lc1*) ∈ *cptn-mod-nest-call* ∧ Γ = Γ

using *cp-lc1 cpn-def* **by** *blast*
then have *f2*: $(n,\Gamma,$ *?m-lc1*$) \in$ *cptn-mod-nest-call* **using** *lc1-not-empty*
by (*metis Cons-lift SmallStepCon.nth-tl cptn-mod-nest-call.CptnModNestSeq1*)
then show *?thesis*
using *f2 f1 lc1-not-empty* **by** (*simp add*: *cpn-def lift-def*)
qed
then have *map-cp'*:$(\Gamma,$*?m-lc1*$) \in$ *cp* Γ (*Seq c1 c2*) *s*
unfolding *cpn-def cp-def*
using *cptn-eq-cptn-mod-nest* **by** *fastforce*
also have *map-assum*:$(\Gamma,$*?m-lc1*$) \in$ *assum* (p,R)
using *sub-assum a10 a11* Γ*1 cp-lc1 lc1-not-empty*
by (*metis SmallStepCon.nth-tl map-is-Nil-conv*)
ultimately have $((\Gamma,$*lc1*$) \in$ *assum*$(p, R))$
using Γ*1 assum-map* **using** *assum-map cp-lc1'* **by** *blast*
then have *lc1-comm*:$(\Gamma,$*lc1*$) \in$ *comm*$(G, (q,a))$ *F*
using *a1 cp-lc1* **by** (*meson IntI com-validityn-def contra-subsetD*)
then have *m-lc1-comm*:$(\Gamma,$*?m-lc1*$) \in$ *comm*$(G, (q,a))$ *F*
using *map-cp' map-assum comm-map cp-lc1'* **by** *fastforce*
then have *i-step*:$\Gamma\vdash_c(l!(i-1)) \rightarrow (l!i)$
proof $-$
have $\Gamma\vdash_c(l!(i-1)) \rightarrow_{ce} (l!(i))$
proof $-$
have *f1*: $\forall n\ na.\ \neg\ n < na \vee Suc\ (na - Suc\ n) = na - n$
by (*meson Suc-diff-Suc*)
have *map* (*lift c2*) *lc1* $\neq$ []
by (*metis lc1-not-empty map-is-Nil-conv*)
then have *f2*: $0 <$ *length* (*map* (*lift c2*) *lc1*)
by (*meson length-greater-0-conv*)
then have *length* (*map* (*lift c2*) *lc1*) $- 1 + 1 <$ *length* (*map* (*lift c2*) *lc1* @ *lc2*)
using *f2 lc2-not-empty* **by** *simp*
then show *?thesis*
using *f2 f1*
proof $-$
have $0 < i$
using *f2 i-map* **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *One-nat-def Suc-diff-1 a0 add.right-neutral add-Suc-right cp cptn-tran-ce-i*)
qed
qed
moreover have $\neg\Gamma\vdash_c(l!(i-1)) \rightarrow_e (l!i)$
using *li last-m-lc1*
by (*metis* (*no-types*, *lifting*) *env-c-c' seq-and-if-not-eq*(*4*))
ultimately show *?thesis* **using** *step-ce-elim-cases* **by** *blast*
qed
then have *step*:$\Gamma\vdash_c($*Seq* (*fst* (*last lc1*)) *c2*,*s2*$) \rightarrow ($*c2*, *s2*$)$
using *last-m-lc1 li* **by** *fastforce*
then obtain *s2'* **where**

last-lc1:*fst* (*last lc1*) = *Skip* ∨
fst (*last lc1*) = *Throw* ∧ (*s2* = *Normal s2'*)
using *seq-skip-throw* **by** *blast*
have *final*:*final* (*last lc1*)
using *last-lc1 l-is* **unfolding** *final-def* **by** *auto*

have *normal-last*:*fst* (*last lc1*) = *Skip* ∧ *snd* (*last lc1*) ∈ *Normal* ‘ *q* ∨
fst (*last lc1*) = *Throw* ∧ *snd* (*last lc1*) ∈ *Normal* ‘ (*a*)
proof −
have *snd* (*last lc1*) ∉ *Fault* ‘ *F*
using *i-not-fault l-is li* **by** *auto*
then show *?thesis*
using *final comm-dest2 lc1-comm* **by** *blast*
qed
obtain *s2'* **where** *lastlc1-normal*:*snd* (*last lc1*) = *Normal s2'*
using *normal-last* **by** *blast*
then have *Normals2*:*s2* = *Normal s2'* **by** (*simp add*: *l-is*)
have *Gs2'*:(*Normal s2'*, *Normal s2'*)∈*G* **using** *a5* **by** *auto*
have *concl*:
(∀ *i*. *Suc i*<*length l* ⟶
$\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)
proof−
{ **fix** *k*
assume *a00*:*Suc k*<*length l* **and**
a21:$\Gamma\vdash_c$(*l*!*k*) → (*l*!(*Suc k*))
have *i-m-l*:∀ *j* <*i* . *l*!*j* = *?m-lc1*!*j*
proof −
have *map* (*lift c2*) *lc1* ≠ []
by (*meson lc1-not-empty list.map-disc-iff*)
then show *?thesis*
using *cp-lc1 i-map length-c1-map* **by** (*fastforce simp*:*nth-append*)

qed
have (*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) ∈ *G*
proof (*cases Suc k*< *i*)
case *True*
then have *a11'*: $\Gamma\vdash_c$(*?m-lc1*!*k*) → (*?m-lc1*!(*Suc k*))
using *a11 i-m-l True*
proof −
have ∀ *n na*. ¬ *0* < *n* − *Suc na* ∨ *na* < *n*
using *diff-Suc-eq-diff-pred zero-less-diff* **by** *presburger*
then show *?thesis* **using** *True a21 i-m-l* **by** *force*
qed
have *Suc k* < *length ?m-lc1* **using** *True i-map length-c1-map* **by** *metis*
then have (*snd*(*?m-lc1*!*k*), *snd*(*?m-lc1*!(*Suc k*))) ∈ *G*
using *a11' last-mcl1-not-F m-lc1-comm True i-map length-c1-map*
comm-dest1[*of* Γ]
by *blast*

```
      thus ?thesis using i-m-l using True by fastforce
    next
      case False
      have (Suc k=i) ∨ (Suc k>i) using False by auto
      thus ?thesis
      proof
      { assume suck:(Suc k=i)
      then have k:k=i−1 by auto
        then show (snd (l!k), snd (l!Suc k)) ∈ G
        proof −
          have snd (l!Suc k) = Normal s2'
            using Normals2 suck li by auto
          moreover have snd (l ! k) = Normal s2'
            using Normals2 k last-m-lc1 by fastforce
          moreover have ∃p. p ∈ G
            by (meson  case-prod-conv mem-Collect-eq Gs2')
          ultimately show ?thesis using suck k Normals2
            using Gs2'  by force
        qed
      }
      next
      {
        assume a001:Suc k>i
        then have k:k≥i by fastforce
        then obtain k' where k':k=i+k'
          using add.commute le-Suc-ex by blast
        {assume throw:c2=Throw ∧ fst (last lc1) = Throw
         then have s2-in:s2' ∈ a
           using Normals2 i-map normal-last li lastlc1-normal
           using image-iff snd-conv xstate.inject(1) by auto

         then have ∀k. k≥i ∧ (Suc k < length l) ⟶
                  ¬(Γ⊢c(l!k)  → (l!(Suc k)))
          using Normals2 li lastlc1-normal a21 a001 a00 a4
                a0 throw env-tran-right only-one-component-tran-j snd-conv
           by (metis  cp env-tran-right)
      then have ?thesis using a21 a001 k a00 by blast
        }  note left=this
        {assume ¬(c2=Throw ∧ fst (last lc1) = Throw)
         then have fst (last lc1) = Skip
           using last-m-lc1 last-lc1
           by (metis step a0 l-is li prod.collapse stepc-Normal-elim-cases(11)
stepc-Normal-elim-cases(5))
         then have s2-normal:s2 ∈ Normal ` q
           using normal-last lastlc1-normal Normals2
           by fastforce
         have length-lc2:length l=i+length lc2
              using i-map cp-lc1 by fastforce
         have (Γ,lc2) ∈  assum (q,R)
```

proof –
have *left*:*snd* (*lc2*!*0*) ∈ *Normal* ‘ *q*
using *li lc2-l s2-normal lc2-not-empty* **by** *fastforce*
{
fix *j*
assume *j-len*:*Suc j*<*length lc2* **and**
j-step:$\Gamma\vdash_c$(*lc2*!*j*) $\rightarrow_e$ (*lc2*!(*Suc j*))

then have *suc-len*:*Suc* (*i* + *j*)<*length l* **using** *j-len length-lc2*
by *fastforce*
also then have $\Gamma\vdash_c$(*l*!(*i*+*j*)) $\rightarrow_e$ (*l*! (*Suc* (*i*+ *j*)))
using *lc2-l j-step j-len* **by** *fastforce*
ultimately have (*snd*(*lc2*!*j*), *snd*(*lc2*!(*Suc j*))) ∈ *R*
using *assum suc-len lc2-l j-len cp* **by** *fastforce*
}
then show *?thesis* **using** *left*
unfolding *assum-def* **by** *fastforce*
qed
also have (Γ,*lc2*) ∈ *cpn n* Γ *c2 s2*
using *cp-lc1 i-map l-is last-conv-nth lc1-not-empty* **by** *fastforce*

ultimately have *comm-lc2*:(Γ,*lc2*) ∈ *comm* (*G*, (*r*,*a*)) *F*
using *a3* **unfolding** *com-validityn-def* **by** *auto*
have *lc2-last-f*:*snd* (*last lc2*)∉ *Fault* ‘ *F*
using *lc2-l lc2-not-empty l-f cp-lc1* **by** *fastforce*
have *suck'*:*Suc k'* < *length lc2*
using *k' a00 length-lc2* **by** *arith*
moreover then have $\Gamma\vdash_c$(*lc2*!*k'*) → (*lc2*!(*Suc k'*))
using *k' lc2-l a21* **by** *fastforce*
ultimately have (*snd* (*lc2*! *k'*), *snd* (*lc2* ! *Suc k'*)) ∈ *G*
using *comm-lc2 lc2-last-f comm-dest1*[*of* Γ *lc2 G r a F k'*]
by *blast*
then have *?thesis* **using** *suck' lc2-l k'* **by** *fastforce*
}
then show *?thesis* **using** *left* **by** *auto*
}
qed
qed
} **thus** *?thesis* **by** *auto*
qed note *left*=*this*
have *right*:(*final* (*last l*) ⟶
((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ‘ *r*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ‘ *a*))
proof –
{ assume *final-l*:*final* (*last l*)
have *eq-last-lc2-l*:*last l*=*last lc2* **by** (*simp add*: *cp-lc1 lc2-not-empty*)
then have *final-lc2*:*final* (*last lc2*) **using** *final-l* **by** *auto*

{
  assume *lst-lc1-throw*:*fst* (*last lc1*) = *Throw*
  then have *c2-throw*:*c2* = *Throw*
    using *lst-lc1-throw step lastlc1-normal stepc-elim-cases-Seq-skip-c2*
    by *fastforce*
  have *s2-a*:*s2* ∈ *Normal* ' (*a*)
    using *normal-last*
    by (*simp add*: *lst-lc1-throw l-is*)
  have *all-ev*:∀ *k*<*length l* − *1*. *k*≥*i* ∧ (*Suc k* < *length l*) ⟶
      $\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k))$
  proof −
    have *s2-in*:*s2'* ∈ *a*
      using *Normals2 i-map normal-last li lastlc1-normal*
      using *image-iff snd-conv xstate.inject*(*1*) *lst-lc1-throw* **by** *auto*
    then have ∀ *k*. *k*≥*i* ∧ (*Suc k* < *length l*) ⟶
      $\neg(\Gamma\vdash_c(l!k) \rightarrow (l!(Suc\ k)))$
      using *Normals2 li lastlc1-normal a4*
        a0 c2-throw env-tran-right only-one-component-tran-j snd-conv
      by (*metis cp env-tran-right*)
  thus *?thesis* **by** (*metis Suc-eq-plus1 cp cptn-tran-ce-i step-ce-elim-cases*)

  qed
  then have *Throw*:*fst* (*l*!(*length l* − *1*)) = *Throw*
  using *cp c2-throw a0 cptn-i-env-same-prog*[*of* Γ *l* ((*length l*)−*1*) *i*]
    by *fastforce*
  then have *snd* (*l*!(*length l* − *1*)) ∈ *Normal* ' (*a*) ∧ *fst* (*l*!(*length l* − *1*)) = *Throw*
    using *all-ev a0 s2-a li a4 env-tran-right stability*[*of a R l i* (*length l*) −*1* - Γ] *Throw*
    by (*metis One-nat-def Suc-pred length-greater-0-conv*
        lessI linorder-not-less list.size(*3*)
        not-less0 not-less-eq-eq snd-conv)
  then have ((*fst* (*last l*) = *Skip* ∧
      snd (*last l*) ∈ *Normal* ' *r*)) ∨
      (*fst* (*last l*) = *Throw* ∧
      snd (*last l*) ∈ *Normal* ' (*a*))
  using *a0* **by** (*metis last-conv-nth list.size*(*3*) *not-less0*)
} **note** *left* = *this*
{ **assume** *fst* (*last lc1*) = *Skip*
  then have *s2-normal*:*s2* ∈ *Normal* ' *q*
    using *normal-last lastlc1-normal Normals2*
    by *fastforce*
  have *length-lc2*:*length l*=*i*+*length lc2*
      using *i-map cp-lc1* **by** *fastforce*
  have (Γ,*lc2*) ∈ *assum* (*q*,*R*)
  proof −
    have *left*:*snd* (*lc2*!*0*) ∈ *Normal* ' *q*
      using *li lc2-l s2-normal lc2-not-empty* **by** *fastforce*
    {

fix j
assume *j-len*:*Suc* j<*length lc2* **and**
j-step:$\Gamma\vdash_c(lc2!j) \rightarrow_e (lc2!(Suc\ j))$
then have *suc-len*:*Suc* $(i + j)$<*length l* **using** *j-len length-lc2*
by *fastforce*
also then have $\Gamma\vdash_c(l!(i+j)) \rightarrow_e (l!\ (Suc\ (i+ j)))$
using *lc2-l j-step j-len* **by** *fastforce*
ultimately have $(snd(lc2!j),\ snd(lc2!(Suc\ j))) \in R$
using *assum suc-len lc2-l j-len cp* **by** *fastforce*
}
then show *?thesis* **using** *left*
unfolding *assum-def* **by** *fastforce*
qed
also have $(\Gamma,lc2) \in$ *cpn n* Γ *c2 s2*
using *cp-lc1 i-map l-is last-conv-nth lc1-not-empty* **by** *fastforce*
ultimately have *comm-lc2*:$(\Gamma,lc2) \in$ *comm* $(G,\ (r,a))$ F
using *a3* **unfolding** *com-validityn-def* **by** *auto*
have *lc2-last-f*:*snd* (*last lc2*)$\notin$ *Fault* ‘ *F*
using *lc2-l lc2-not-empty l-f cp-lc1* **by** *fastforce*
then have ((*fst* (*last lc2*) = *Skip* $\wedge$
snd (*last lc2*) $\in$ *Normal* ‘ *r*)) $\vee$
(*fst* (*last lc2*) = *Throw* $\wedge$
snd (*last lc2*) $\in$ *Normal* ‘ *a*)
using *final-lc2 comm-lc2* **unfolding** *comm-def* **by** *auto*
then have ((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *r*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *a*)
using *eq-last-lc2-l* **by** *auto*
}
then have ((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *r*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *a*)
using *left* **using** *last-lc1* **by** *auto*
} thus *?thesis* **by** *auto* **qed**
thus *?thesis* **using** *left l-f* Γ*1* **unfolding** *comm-def* **by** *force*
qed
} thus *?thesis* **using** Γ*1* **unfolding** *comm-def* **by** *auto* **qed**
} thus *?thesis* **by** *auto* **qed**
} thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Catch-env-P*:**assumes** *a0*:$\Gamma\vdash_c(Catch\ P\ Q,s) \rightarrow_e (Catch\ P\ Q,t)$
shows $\Gamma\vdash_c(P,s) \rightarrow_e (P,t)$
using *a0*
by (*metis env-not-normal-s snormal-enviroment*)

lemma *map-catch-eq-state*:

assumes
  $a0$:$(\Gamma,l1) \in (cp\ \Gamma\ (Catch\ c1\ c2)\ s)$ **and**
  $a1$:$(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
  $a2$:$l1=map\ (lift\text{-}catch\ c2)\ l2$
shows
  $\forall i<length\ l1.\ snd\ (l1!i) = snd\ (l2!i)$
using *a0 a1 a2* **unfolding** *cp-def*
by (*simp add*: *snd-lift-catch*)

lemma *map-eq-catch-c*:
assumes
  $a0$:$(\Gamma,l1) \in (cp\ \Gamma\ (Catch\ c1\ c2)\ s)$ **and**
  $a1$:$(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
  $a2$:$l1=map\ (lift\text{-}catch\ c2)\ l2$
shows
  $\forall i<length\ l1.\ fst\ (l1!i) = Catch\ (fst\ (l2!i))\ c2$
proof −
  {**fix** *i*
  assume $a3$:$i<length\ l1$
  have $fst\ (l1!i) = Catch\ (fst\ (l2!i))\ c2$
  using *a0 a1 a2 a3* **unfolding** *lift-catch-def*
    by (*simp add*: *case-prod-unfold*)
  }**thus** *?thesis* **by** *auto*
qed

lemma *same-env-catch-c*:
assumes
  $a0$:$(\Gamma,l1) \in (cp\ \Gamma\ (Catch\ c1\ c2)\ s)$ **and**
  $a1$:$(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
  $a2$:$l1=map\ (lift\text{-}catch\ c2)\ l2$
shows
$\forall i.\ Suc\ i<length\ l2 \longrightarrow \Gamma\vdash_c(l2!i)\ \rightarrow_e (l2!(Suc\ i)) =$
      $\Gamma\vdash_c(l1!i)\ \rightarrow_e (l1!(Suc\ i))$
proof −
  have $a0a$:$(\Gamma,l1) \in cptn \wedge l1!0 = ((Catch\ c1\ c2),s)$
    using *a0* **unfolding** *cp-def* **by** *blast*
  have $a1a$: $(\Gamma,l2) \in cptn \wedge l2!0 = (c1,s)$
    using *a1* **unfolding** *cp-def* **by** *blast*
  {
    fix *i*
    assume $a3$:$Suc\ i<\ length\ l2$
    have $\Gamma\vdash_c(l2!i)\ \rightarrow_e (l2!(Suc\ i)) =$
        $\Gamma\vdash_c(l1!i)\ \rightarrow_e (l1!(Suc\ i))$
    proof
    {
      assume $a4$:$\Gamma\vdash_c\ l2\ !\ i \rightarrow_e l2\ !\ Suc\ i$
      obtain *c1i s1i c1si s1si* **where** $l1prod$:$l1\ !\ i=(c1i,s1i) \wedge l1!Suc\ i = (c1si,s1si)$
        by *fastforce*
      obtain *c2i s2i c2si s2si* **where** $l2prod$:$l2\ !\ i=(c2i,s2i) \wedge l2!Suc\ i = (c2si,s2si)$

by *fastforce*
then have $c1i = (Catch\ c2i\ c2) \wedge c1si = (Catch\ c2si\ c2)$
using *a0 a1 a2 a3 a4 l1prod*
by (*simp add*: *lift-catch-def*)
also have $s2i{=}s1i \wedge s2si{=}s1si$
using *a0 a1 a4 a2 a3 l2prod l1prod*
by (*simp add*: *lift-catch-def*)
ultimately show $\Gamma \vdash_c l1\ !\ i \rightarrow_e (l1\ !\ Suc\ i)$
using *a4 l1prod l2prod*
by (*metis Env-n env-c-c′ env-not-normal-s step-e.Env*)
}
{
assume $a4{:}\Gamma \vdash_c l1\ !\ i \rightarrow_e l1\ !\ Suc\ i$
obtain *c1i s1i c1si s1si* **where** $l1prod{:}l1\ !\ i{=}(c1i,s1i) \wedge l1!Suc\ i = (c1si,s1si)$
by *fastforce*
obtain *c2i s2i c2si s2si* **where** $l2prod{:}l2\ !\ i{=}(c2i,s2i) \wedge l2!Suc\ i = (c2si,s2si)$
by *fastforce*
then have $c1i = (Catch\ c2i\ c2) \wedge c1si = (Catch\ c2si\ c2)$
using *a0 a1 a2 a3 a4 l1prod*
by (*simp add*: *lift-catch-def*)
also have $s2i{=}s1i \wedge s2si{=}s1si$
using *a0 a1 a4 a2 a3 l2prod l1prod*
by (*simp add*: *lift-catch-def*)
ultimately show $\Gamma \vdash_c l2\ !\ i \rightarrow_e (l2\ !\ Suc\ i)$
using *a4 l1prod l2prod*
by (*metis Env-n LanguageCon.com.inject*(*9*) *env-c-c′ env-not-normal-s step-e.Env*)
}
qed
}
thus *?thesis* **by** *auto*
qed

lemma *same-comp-catch-c*:
assumes
$a0{:}(\Gamma,l1) \in (cp\ \Gamma\ (Catch\ c1\ c2)\ s)$ **and**
$a1{:}(\Gamma,l2) \in (cp\ \Gamma\ c1\ s)$ **and**
$a2{:}l1{=}map\ (lift\text{-}catch\ c2)\ l2$
shows
$\forall i.\ Suc\ i{<}length\ l2 \longrightarrow \Gamma \vdash_c (l2!i) \rightarrow (l2!(Suc\ i)) =$
$\Gamma \vdash_c (l1!i) \rightarrow (l1!(Suc\ i))$
proof −
have $a0a{:}(\Gamma,l1) \in cptn \wedge l1!0 = ((Catch\ c1\ c2),s)$
using *a0* **unfolding** *cp-def* **by** *blast*
have $a1a{:}\ (\Gamma,l2) \in cptn \wedge l2!0 = (c1,s)$
using *a1* **unfolding** *cp-def* **by** *blast*
{
fix *i*

assume *a3*:*Suc i*< *length l2*
have Γ⊢$_c$(*l2*!*i*) → (*l2*!(*Suc i*)) =
Γ⊢$_c$(*l1*!*i*) → (*l1*!(*Suc i*))
proof
{
assume *a4*:Γ⊢$_c$ *l2* ! *i* → *l2* ! *Suc i*
obtain *c1i s1i c1si s1si* **where** *l1prod*:*l1* ! *i*=(*c1i*,*s1i*) ∧ *l1*!*Suc i* = (*c1si*,*s1si*)
by *fastforce*
obtain *c2i s2i c2si s2si* **where** *l2prod*:*l2* ! *i*=(*c2i*,*s2i*) ∧ *l2*!*Suc i* = (*c2si*,*s2si*)
by *fastforce*
then have *c1i* = (*Catch c2i c2*) ∧ *c1si* = (*Catch c2si c2*)
using *a0 a1 a2 a3 a4 map-eq-catch-c l1prod*
by (*simp add*: *lift-catch-def*)
also have *s2i*=*s1i* ∧ *s2si*=*s1si*
using *a0 a1 a4 a2 a3 l2prod map-eq-state l1prod*
by (*simp add*: *lift-catch-def*)
ultimately show Γ⊢$_c$ *l1* ! *i* → (*l1* ! *Suc i*)
using *a4 l1prod l2prod*
by (*simp add*: *Catchc*)
}
{
assume *a4*:Γ⊢$_c$ *l1* ! *i* → *l1* ! *Suc i*
obtain *c1i s1i c1si s1si* **where** *l1prod*:*l1* ! *i*=(*c1i*,*s1i*) ∧ *l1*!*Suc i* = (*c1si*,*s1si*)
by *fastforce*
obtain *c2i s2i c2si s2si* **where** *l2prod*:*l2* ! *i*=(*c2i*,*s2i*) ∧ *l2*!*Suc i* = (*c2si*,*s2si*)
by *fastforce*
then have *c1i* = (*Catch c2i c2*) ∧ *c1si* = (*Catch c2si c2*)
using *a0 a1 a2 a3 a4 l1prod*
by (*simp add*: *lift-catch-def*)
also have *s2i*=*s1i* ∧ *s2si*=*s1si*
using *a0 a1 a4 a2 a3 l2prod l1prod*
by (*simp add*: *lift-catch-def*)
ultimately show Γ⊢$_c$ *l2* ! *i* → (*l2* ! *Suc i*)
using *a4 l1prod l2prod stepc-elim-cases-Catch-Catch Catch-not-c*
by (*metis* (*no-types*))
}
qed
}
thus *?thesis* **by** *auto*
qed

lemma *assum-map-catch*:
assumes
a0:(Γ,*l1*) ∈ (*cp* Γ (*Catch c1 c2*) *s*) ∧ ((Γ,*l1*) ∈ *assum*(*p*, *R*)) **and**
a1:(Γ,*l2*) ∈ (*cp* Γ *c1 s*) **and**
a2:*l1*=*map* (*lift-catch c2*) *l2*
shows
((Γ,*l2*) ∈ *assum*(*p*, *R*))
proof −

have *a3*: $\forall$ *i*. *Suc i*<*length l2* $\longrightarrow$ $\Gamma\vdash_c$(*l2*!*i*) $\rightarrow_e$ (*l2*!(*Suc i*)) =
$\Gamma\vdash_c$(*l1*!*i*) $\rightarrow_e$ (*l1*!(*Suc i*))
using *a0 a1 a2 same-env-catch-c* **by** *fastforce*
have *pair-Γl1*:*fst* (Γ,*l1*) = Γ ∧ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
have *pair-Γl2*:*fst* (Γ,*l2*) = Γ ∧ *snd* (Γ,*l2*) = *l2* **by** *fastforce*
have *drop-k-s*:*l2*!*0* = (*c1*,*s*) **using** *a1 cp-def* **by** *blast*
have *eq-length*:*length l1* = *length l2* **using** *a2* **by** *auto*
obtain *s'* **where** *normal-s*:*s* = *Normal s'*
using *a0* **unfolding** *cp-def assum-def* **by** *fastforce*
then have *p1*:*s'*∈*p* **using** *a0* **unfolding** *cp-def assum-def* **by** *fastforce*
show *?thesis*
proof −
let *?c*= (Γ,*l2*)
have *l*:*snd*((*snd ?c*!*0*)) ∈ *Normal* ‘ (*p*)
using *p1 drop-k-s a1 normal-s* **unfolding** *cp-def* **by** *auto*
{**fix** *i*
assume *a00*:*Suc i*<*length* (*snd ?c*)
assume *a11*:(*fst ?c*)$\vdash_c$((*snd ?c*)!*i*) $\rightarrow_e$ ((*snd ?c*)!(*Suc i*))
have (*snd*((*snd ?c*)!*i*), *snd*((*snd ?c*)!(*Suc i*))) ∈ *R*
using *a0 a1 a2 a3 map-catch-eq-state* **unfolding** *assum-def*
using *a00 a11 eq-length* **by** *fastforce*
} **thus** (Γ, *l2*) ∈ *assum* (*p*, *R*)
using *l* **unfolding** *assum-def* **by** *fastforce*
qed
qed

lemma *comm-map'-catch*:
assumes
a0:(Γ,*l1*) ∈ (*cp* Γ (*Catch c1 c2*) *s*) **and**
a1:(Γ,*l2*) ∈ (*cp* Γ *c1 s*) ∧ (Γ, *l2*)∈ *comm*(*G*, (*q*,*a*)) *F* **and**
a2:*l1*=*map* (*lift-catch c2*) *l2*
shows
snd (*last l1*) ∉ *Fault* ‘ *F* $\longrightarrow$(*Suc k* < *length l1* $\longrightarrow$
$\Gamma\vdash_c$(*l1*!*k*) $\rightarrow$ (*l1*!(*Suc k*)) $\longrightarrow$
(*snd*(*l1*!*k*), *snd*(*l1*!(*Suc k*))) ∈ *G*) ∧
(*fst* (*last l1*) = (*Catch c c2*) ∧ *final* (*c*, *snd* (*last l1*)) $\longrightarrow$
(*fst* (*last l1*) = (*Catch Skip c2*) ∧
(*snd* (*last l1*) ∈ *Normal* ‘ *q*) ∨
(*fst* (*last l1*) = (*Catch Throw c2*) ∧
snd (*last l1*) ∈ *Normal* ‘ (*a*))))

proof −
have *a3*:$\forall$ *i*. *Suc i*<*length l2* $\longrightarrow$ $\Gamma\vdash_c$(*l2*!*i*) $\rightarrow$ (*l2*!(*Suc i*)) =
$\Gamma\vdash_c$(*l1*!*i*) $\rightarrow$ (*l1*!(*Suc i*))
using *a0 a1 a2 same-comp-catch-c*
by *fastforce*
have *pair-Γl1*:*fst* (Γ,*l1*) = Γ ∧ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
have *pair-Γl2*:*fst* (Γ,*l2*) = Γ ∧ *snd* (Γ,*l2*) = *l2* **by** *fastforce*
have *drop-k-s*:*l2*!*0* = (*c1*,*s*) **using** *a1 cp-def* **by** *blast*

have *eq-length*:*length l1* = *length l2* **using** *a2* **by** *auto*
have *len0*:*length l2*>*0* **using** *a1* **unfolding** *cp-def*
  using *cptn.simps* **by** *fastforce*
then have *len0*:*length l1*>*0* **using** *eq-length* **by** *auto*
then have *l1-not-empty*:*l1*≠[] **by** *auto*
then have *l2-not-empty*:*l2* ≠ [] **using** *a2* **by** *blast*
 have *last-lenl1*:*last l1* = *l1*!((*length l1*) −*1*)
   using *last-conv-nth l1-not-empty* **by** *auto*
have *last-lenl2*:*last l2* = *l2*!((*length l2*) −*1*)
   using *last-conv-nth l2-not-empty* **by** *auto*
have *a03*:*snd* (*last l2*) ∉ *Fault* ' *F* ⟶(∀ *i ns ns'*.
    Suc i<*length* (*snd* (Γ, *l2*)) ⟶
      fst (Γ, *l2*)⊢$_c$((*snd* (Γ, *l2*))!*i*) → ((*snd* (Γ, *l2*))!(*Suc i*)) ⟶

     (*snd*((*snd* (Γ, *l2*))!*i*), *snd*((*snd* (Γ, *l2*))!(*Suc i*))) ∈ *G*) ∧
    (*final* (*last* (*snd* (Γ, *l2*))) ⟶
     ((*fst* (*last* (*snd* (Γ, *l2*))) = *Skip* ∧
      snd (*last* (*snd* (Γ, *l2*))) ∈ *Normal* ' *q*)) ∨
     (*fst* (*last* (*snd* (Γ, *l2*))) = *Throw* ∧
      snd (*last* (*snd* (Γ, *l2*))) ∈ *Normal* ' (*a*)))
using *a1* **unfolding** *comm-def* **by** *fastforce*
show *?thesis* **unfolding** *comm-def*
proof −
{ **fix** *k ns ns'*
 assume *a00a*:*snd* (*last l1*) ∉ *Fault* ' *F*
 assume *a00*:*Suc k* < *length l1*
 then have *k* ≤ *length l1* **using** *a2* **by** *fastforce*
 have *a00*:*Suc k* < *length l2* **using** *eq-length a00* **by** *fastforce*
 then have *a00a*:*snd* (*last l2*) ∉ *Fault* ' *F*
 proof−
  have *snd* (*l1*!((*length l1*) −*1*)) = *snd* (*l2*!((*length l2*) −*1*))
   using *a2 a1 a0 map-catch-eq-state eq-length l2-not-empty last-snd*
   by *fastforce*
  then have *snd*(*last l2*) = *snd* (*last l1*)
   using *last-lenl1 last-lenl2* **by** *auto*
  thus *?thesis* **using** *a00a* **by** *auto*
 qed
 then have *snd* (*last l1*) ∉ *Fault* ' *F* ⟶Γ⊢$_c$(*l1*!*k*) → (*l1*!(*Suc k*)) ⟶
  (*snd*((*snd* (Γ, *l1*))!*k*), *snd*((*snd* (Γ, *l1*))!(*Suc k*))) ∈ *G*
 using *pair-*Γ*l1 pair-*Γ*l2 a00 a03 a3 eq-length a00a*
  by (*metis Suc-lessD a0 a1 a2 map-catch-eq-state*)
} **note** *l*=*this*
{
 assume *a00*: *fst* (*last l1*) = (*Catch c c2*) ∧ *final* (*c*, *snd* (*last l1*)) **and**
    a01:*snd* (*last* (*l1*)) ∉ *Fault* ' *F*
 then have *c*:*c*=*Skip* ∨ *c* = *Throw*
  unfolding *final-def* **by** *auto*
 then have *fst-last-l2*:*fst* (*last l2*) = *c*
  using *last-lenl1 a00 l1-not-empty eq-length len0 a2 last-conv-nth last-lift-catch*

by *fastforce*
also have *last-eq*:*snd* (*last l2*) = *snd* (*last l1*)
using *l2-not-empty a2 last-conv-nth last-lenl1 last-snd-catch*
by *fastforce*
ultimately have *final* (*fst* (*last l2*),*snd* (*last l2*))
using *a00* **by** *auto*
then have *final* (*last l2*) **by** *auto*
also have *snd* (*last* (*l2*)) $\notin$ *Fault* ‘ *F*
using *last-eq a01* **by** *auto*
ultimately have (*fst* (*last l2*)) = *Skip* $\wedge$
snd (*last l2*) $\in$ *Normal* ‘ *q* $\vee$
(*fst* (*last l2*) = *Throw* $\wedge$
snd (*last l2*) $\in$ *Normal* ‘ (*a*))
using *a03* **by** *auto*
then have (*fst* (*last l1*) = (*Catch Skip c2*) $\wedge$
snd (*last l1*) $\in$ *Normal* ‘ *q*) $\vee$
(*fst* (*last l1*) = (*Catch Throw c2*) $\wedge$
snd (*last l1*) $\in$ *Normal* ‘ (*a*))
using *last-eq fst-last-l2 a00* **by** *force*
}
thus *?thesis* **using** *l* **by** *auto* **qed**
qed

lemma *comm-map′′-catch*:
assumes
a0:(Γ,*l1*) $\in$ (*cp* Γ (*Catch c1 c2*) *s*) **and**
a1:(Γ,*l2*) $\in$ (*cp* Γ *c1 s*) $\wedge$ (Γ, *l2*)$\in$ *comm*(*G*, (*q*,*a*)) *F* **and**
a2:*l1*=*map* (*lift-catch c2*) *l2*
shows
snd (*last l1*) $\notin$ *Fault* ‘ *F* $\longrightarrow$ ((*Suc k* < *length l1* $\longrightarrow$
$\Gamma\vdash_c$(*l1*!*k*) $\rightarrow$ (*l1*!(*Suc k*)) $\longrightarrow$
(*snd*(*l1*!*k*), *snd*(*l1*!(*Suc k*))) $\in$ *G*) $\wedge$
(*final* (*last l1*) $\longrightarrow$
(*fst* (*last l1*) = *Skip* $\wedge$
(*snd* (*last l1*) $\in$ *Normal* ‘ *r*) $\vee$
(*fst* (*last l1*) = *Throw* $\wedge$
snd (*last l1*) $\in$ *Normal* ‘ *a*))))

proof −
have *a3*:$\forall$ *i*. *Suc i*<*length l2* $\longrightarrow$ $\Gamma\vdash_c$(*l2*!*i*) $\rightarrow$ (*l2*!(*Suc i*)) =
$\Gamma\vdash_c$(*l1*!*i*) $\rightarrow$ (*l1*!(*Suc i*))
using *a0 a1 a2 same-comp-catch-c*
by *fastforce*
have *pair-*Γ*l1*:*fst* (Γ,*l1*) = Γ $\wedge$ *snd* (Γ,*l1*) = *l1* **by** *fastforce*
have *pair-*Γ*l2*:*fst* (Γ,*l2*) = Γ $\wedge$ *snd* (Γ,*l2*) = *l2* **by** *fastforce*
have *drop-k-s*:*l2*!*0* = (*c1*,*s*) **using** *a1 cp-def* **by** *blast*
have *eq-length*:*length l1* = *length l2* **using** *a2* **by** *auto*
have *len0*:*length l2*>*0* **using** *a1* **unfolding** *cp-def*

```
    using cptn.simps  by fastforce
then have len0:length l1>0 using eq-length by auto
then have l1-not-empty:l1≠[] by auto
then have l2-not-empty:l2 ≠ [] using a2 by blast
have last-lenl1:last l1 = l1!((length l1) −1)
      using last-conv-nth l1-not-empty by auto
have last-lenl2:last l2 = l2!((length l2) −1)
     using last-conv-nth l2-not-empty by auto
have a03:snd (last l2) ∉ Fault ‘ F ⟶(∀ i ns ns'.
            Suc i<length (snd (Γ, l2)) ⟶
                  fst (Γ, l2)⊢c((snd (Γ, l2))!i)  → ((snd (Γ, l2))!(Suc i)) ⟶

              (snd((snd (Γ, l2))!i), snd((snd (Γ, l2))!(Suc i))) ∈ G) ∧
            (final (last (snd (Γ, l2)))  ⟶
               ((fst (last (snd (Γ, l2))) = Skip ∧
                 snd (last (snd (Γ, l2))) ∈ Normal ‘ q)) ∨
               (fst (last (snd (Γ, l2))) = Throw ∧
                 snd (last (snd (Γ, l2))) ∈ Normal ‘ (a)))
using a1 unfolding comm-def by fastforce
show ?thesis unfolding comm-def
proof −
{ fix k ns ns'
  assume a00a:snd (last l1) ∉ Fault ‘ F
  assume a00:Suc k < length l1
  then have k ≤ length l1 using a2 by fastforce
  have a00:Suc k < length l2 using eq-length a00 by fastforce
  then have a00a:snd (last l2) ∉ Fault ‘ F
  proof−
    have snd (l1!((length l1) −1)) = snd (l2!((length l2) −1))
      using a2 a1 a0  map-catch-eq-state eq-length l2-not-empty last-snd
      by fastforce
    then have snd(last l2) = snd (last l1)
      using last-lenl1 last-lenl2 by auto
    thus ?thesis using a00a by auto
  qed
  then have Γ⊢c(l1!k)  → (l1!(Suc k)) ⟶
      (snd((snd (Γ, l1))!k), snd((snd (Γ, l1))!(Suc k))) ∈ G
     using  pair-Γl1 pair-Γl2 a00  a03 a3  eq-length a00a
    by (metis (no-types,lifting) a2 Suc-lessD nth-map snd-lift-catch)
  } note l= this
  {
   assume a00: final (last l1)
   then have c:fst (last l1)=Skip ∨ fst (last l1) = Throw
     unfolding final-def by auto
   moreover have fst (last l1) = Catch (fst (last l2)) c2
     using a2 last-lenl1 eq-length
    proof −
      have last l2 = l2 ! (length l2 − 1)
        using l2-not-empty last-conv-nth by blast
```

then show *?thesis*
by (*metis One-nat-def a2 l2-not-empty last-lenl1 last-lift-catch*)
qed
ultimately have *False* **by** *simp*
} **thus** *?thesis* **using** *l* **by** *auto* **qed**
qed

lemma *comm-map-catch*:
assumes
a0:(Γ,*l1*) $\in$ (*cp* Γ (*Catch c1 c2*) *s*) **and**
a1:(Γ,*l2*) $\in$ (*cp* Γ *c1 s*) $\wedge$ (Γ, *l2*)$\in$ *comm*(*G*, (*q*,*a*)) *F* **and**
a2:*l1*=*map* (*lift-catch c2*) *l2*
shows
(Γ, *l1*)$\in$ *comm*(*G*, (*r*,*a*)) *F*
proof –
{fix *i ns ns'*
have *snd* (*last l1*) $\notin$ *Fault* ' *F* $\longrightarrow$(*Suc i* < *length* (*l1*) $\longrightarrow$
$\Gamma\vdash_c$ (*l1* ! *i*) $\rightarrow$ (*l1* ! (*Suc i*)) $\longrightarrow$
(*snd* (*l1* ! *i*), *snd* (*l1* ! *Suc i*)) $\in$ *G*) $\wedge$
(*SmallStepCon.final* (*last l1*) $\longrightarrow$
fst (*last l1*) = *LanguageCon.com.Skip* $\wedge$
snd (*last l1*) $\in$ *Normal* ' *r* $\vee$
fst (*last l1*) = *LanguageCon.com.Throw* $\wedge$
snd (*last l1*) $\in$ *Normal* ' *a*)
using *comm-map''-catch*[*of* Γ *l1 c1 c2 s l2 G q a F i r*] *a0 a1 a2*
by *fastforce*
} **then show** *?thesis* **using** *comm-def* **unfolding** *comm-def* **by** *force*
qed

lemma *Catch-sound1*:
assumes
a0:(*n*,Γ,*x*)$\in$*cptn-mod-nest-call* **and**
a1:*x*!*0* = ((*Catch P Q*),*s*) **and**
a2:$\forall$ *i*<*length x*. *fst* (*x*!*i*)$\neq$ *Q* **and**
a3:$\neg$ *final* (*last x*) **and**
a4:*env-tran-right* Γ *x rely*
shows
$\exists$*xs*. (Γ,*xs*) $\in$ *cpn n* Γ *P s* $\wedge$ *x* = *map* (*lift-catch Q*) *xs*
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *P s*)
case (*CptnModNestOne n* Γ *C s1*)
then have (Γ, [(*P*,*s*)]) $\in$ *cpn n* Γ *P s* $\wedge$ [(*C*, *s1*)] = *map* (*lift-catch Q*) [(*P*,*s*)]
unfolding *cpn-def lift-catch-def*
by (*simp add*: *cptn-mod-nest-call.CptnModNestOne*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
then have *C*:*C*=*Catch P Q* **unfolding** *lift-catch-def* **by** *fastforce*
have $\exists$*xs*. (Γ, *xs*) $\in$ *cpn n* Γ *P t1* $\wedge$ (*C*, *t1*) # *xsa* = *map* (*lift-catch Q*) *xs*

```
  proof -
    have ((C, t1) # xsa) ! 0 = (Catch P Q, t1) using C by auto
    moreover have ∀ i<length ((C, t1) # xsa). fst (((C, t1) # xsa) ! i) ≠ Q
      using CptnModNestEnv(5) by fastforce
     moreover have ¬ SmallStepCon.final (last ((C, t1) # xsa)) using Cptn-
ModNestEnv(6)
      by fastforce
    ultimately show ?thesis
      using CptnModNestEnv(3) CptnModNestEnv(7) env-tran-tail by blast
  qed
   then obtain xs where hi:(Γ, xs) ∈ cpn n Γ P t1 ∧ (C, t1) # xsa = map
(lift-catch Q) xs
    by fastforce
  have s1-s:s1=s using  CptnModNestEnv unfolding cpn-def by auto
 obtain xsa' where xs:xs=((P,t1)#xsa') ∧ (n,Γ,((P,t1)#xsa'))∈cptn-mod-nest-call
∧
                 (C, t1) # xsa = map (lift-catch Q) ((P,t1)#xsa')
    using hi  unfolding cpn-def by fastforce

   have env-tran:Γ⊢c(P,s1) →e (P,t1) using CptnModNestEnv Catch-env-P by
(metis fst-conv nth-Cons-0)
  then have (n,Γ,(P,s1)#(P,t1)#xsa')∈cptn-mod-nest-call using xs env-tran
    using cptn-mod-nest-call.CptnModNestEnv by blast
  then have (Γ,(P,s1)#(P,t1)#xsa') ∈ cpn n Γ P s
    using cpn-def s1-s by fastforce
 moreover have (C,s1)#(C, t1) # xsa = map (lift-catch Q) ((P,s1)#(P,t1)#xsa')
    using xs C unfolding lift-catch-def by fastforce
  ultimately show ?case by auto
next
  case (CptnModNestSkip)
  thus ?case by (metis SmallStepCon.redex-not-Catch fst-conv nth-Cons-0)
next
  case (CptnModNestThrow)
  thus ?case by (metis SmallStepCon.redex-not-Catch fst-conv nth-Cons-0)
next
  case (CptnModNestCatch1 n Γ P0 sa xsa zs P1)
  then have a1:LanguageCon.com.Catch P Q = LanguageCon.com.Catch P0 P1
    by fastforce
  have f1: sa = s
    using CptnModNestCatch1.prems(1) by force
  have f2: P = P0 ∧ Q = P1 using a1 by auto
  have (n,Γ, (P0, sa) # xsa) ∈ cptn-mod-nest-call
    by (metis CptnModNestCatch1.hyps(1))
  hence (Γ, (P0, sa) # xsa) ∈ cpn n Γ P s
    using f2 f1 by (simp add: cpn-def)
  thus ?case
    using Cons-lift-catch CptnModNestCatch1.hyps(3) a1 by blast
next
 case (CptnModNestCatch2 n Γ P1 sa xsa  ys zs Q1)
```

have *final* (*last* ((*Skip*, *sa*)# *ys*))
proof −
have *cptn*:(*n*, Γ, (*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call*

using *CptnModNestCatch2*(*4*) **by** (*simp add*: *cptn-eq-cptn-mod-set*)
then have *cptn'*:(Γ, (*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*) ∈ *cptn*
using *cptn-eq-cptn-mod-nest* **by** *blast*
moreover have *throw-0*:((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)!*0* = (*Skip*, *snd* (*last* ((*P1*, *sa*) # *xsa*))) ∧ *0* < *length*((*Skip*, *snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)
by *force*
moreover have *last*:*last* ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*) =
((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)!((*length* ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)) − *1*)
using *last-conv-nth* **by** *auto*
moreover have *env-tran*:*env-tran-right* Γ ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*) *rely*
using *CptnModNestCatch2.hyps*(*6*) *CptnModNestCatch2.prems*(*4*) *env-tran-subl* *env-tran-tail* **by** *blast*
ultimately obtain *st'* **where** *fst* (*last* ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)) = *Skip* ∧
snd (*last* ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*)) = *Normal* *st'*
using *CptnModNestCatch2* *zero-skip-all-skip*[*of* Γ ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*) (*length* ((*Skip*,*snd* (*last* ((*P1*, *sa*) # *xsa*))) # *ys*))−*1*]
by (*metis* (*no-types*, *hide-lams*) *SmallStepCon.final-def* *append-Cons* *diff-less* *fst-conv*
last-appendR *list.simps*(*3*) *zero-less-one*)
thus *?thesis* **using** *final-def* **by** (*metis* *fst-conv* *last.simps*)
qed
thus *?case*
by (*metis* (*no-types*, *lifting*) *CptnModNestCatch2.hyps*(*3*) *CptnModNestCatch2.hyps*(*6*)

CptnModNestCatch2.prems(*3*) *SmallStepCon.final-def* *append-is-Nil-conv* *last.simps*
last-appendR *list.simps*(*3*) *prod.collapse*)
next
case (*CptnModNestCatch3* *n* Γ *P0* *sa* *xsa* *sa'* *P1* *ys* *zs*)
then have *P0* = *P* ∧ *P1* = *Q* **by** *auto*
then obtain *i* **where** *zs*:*fst* (*zs*!*i*) = *Q* ∧ (*i*< (*length* *zs*))
using *CptnModNestCatch3*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left'* *fst-conv* *length-Cons* *length-append* *nth-append-length* *zero-less-Suc* *zero-less-diff*)
then have *Suc* *i*< *length* ((*Catch* *P0* *P1*,*Normal* *sa*)#*zs*) **by** *fastforce*
then have *fst* (((*Catch* *P0* *P1*, *Normal* *sa*) # *zs*)!*Suc* *i*) = *Q* **using** *zs* **by** *fastforce*
thus *?case* **using** *CptnModNestCatch3*(*9*) *zs* **by** *auto*
qed (*auto*)

lemma *Catch-sound2*:
assumes
a0:(n,Γ,x)∈*cptn-mod-nest-call* **and**
a1:*x*!*0* = ((*Catch P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:*fst* (*last x*) = *Skip* **and**
a4:*env-tran-right* Γ *x rely*
shows
∃ *xs ys*. (Γ,*xs*) ∈ *cpn n* Γ *P s* ∧ *x* = ((*map* (*lift-catch Q*) *xs*)@((*Skip*,*snd*(*last* *xs*))#*ys*))
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *P s*)
case (*CptnModNestOne n* Γ *C s1*)
thus *?case* **by** *fastforce*
next
case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
then have *C*:*C*=*Catch P Q* **unfolding** *lift-catch-def* **by** *fastforce*
have ∃ *xs ys*. (Γ, *xs*) ∈ *cpn n* Γ *P t1* ∧ (*C*, *t1*) # *xsa* =
map (*lift-catch Q*) *xs*@((*Skip*, *snd*(*last xs*))#*ys*)
proof −
have ((*C*, *t1*) # *xsa*) ! *0* = (*LanguageCon.com.Catch P Q*, *t1*) **using** *C* **by** *auto*
moreover have ∀ *i*<*length* ((*C*, *t1*) # *xsa*). *fst* (((*C*, *t1*) # *xsa*) ! *i*) ≠ *Q*
using *CptnModNestEnv*(*5*) **by** *fastforce*
moreover have *fst* (*last* ((*C*, *t1*) # *xsa*)) = *Skip* **using** *CptnModNestEnv*(*6*)

by *fastforce*
ultimately show *?thesis*
using *CptnModNestEnv*(*3*) *CptnModNestEnv*(*7*) *env-tran-tail* **by** *blast*
qed
then obtain *xs ys* **where** *hi*:(Γ, *xs*) ∈ *cpn n* Γ *P t1* ∧ (*C*, *t1*) # *xsa* = *map* (*lift-catch Q*) *xs*@((*Skip*,*snd*(*last* ((*P*, *t1*)#*xs*)))#*ys*)
by *fastforce*
have *s1-s*:*s1*=*s* **using** *CptnModNestEnv* **unfolding** *cp-def* **by** *auto*
have ∃ *xsa′ ys*. *xs*=((*P*,*t1*)#*xsa′*) ∧ (*n*,Γ,((*P*,*t1*)#*xsa′*))∈*cptn-mod-nest-call* ∧ (*C*, *t1*) # *xsa* = *map* (*lift-catch Q*) ((*P*,*t1*)#*xsa′*)@((*Skip*, *snd*(*last xs*))#*ys*)
using *hi* **unfolding** *cp-def*
proof −
have (*n*,Γ,*xs*)∈*cptn-mod-nest-call* ∧ *xs*!*0* = (*P*,*t1*) **using** *hi* **unfolding** *cpn-def* **by** *fastforce*
moreover then have *xs*≠[] **using** *CptnEmpty calculation* **by** *blast*
moreover obtain *xsa′* **where** *xs*=((*P*,*t1*)#*xsa′*) **using** *SmallStepCon.nth-tl* *calculation* **by** *fastforce*
ultimately show *?thesis*
using *hi* **by** *auto*
qed
then obtain *xsa′ ys* **where** *xs*:*xs*=((*P*,*t1*)#*xsa′*) ∧ (*n*,Γ,((*P*,*t1*)#*xsa′*))∈*cptn-mod-nest-call* ∧ (*C*, *t1*) # *xsa* =
map (*lift-catch Q*) ((*P*,*t1*)#*xsa′*)@((*Skip*,*snd*(*last*

$((P,s1)\#(P,t1)\#xsa')))\#ys)$
by *fastforce*
have *env-tran*:$\Gamma\vdash_c(P,s1) \rightarrow_e (P,t1)$ **using** *CptnModNestEnv Catch-env-P* **by** (*metis fst-conv nth-Cons-0*)
then have $(n,\Gamma,(P,s1)\#(P,t1)\#xsa')\in$ *cptn-mod-nest-call* **using** *xs env-tran CptnEnv*
by (*simp add*: *cptn-mod-nest-call.CptnModNestEnv*)
then have $(\Gamma,(P,s1)\#(P,t1)\#xsa') \in$ *cpn n* Γ *P s*
using *cpn-def s1-s* **by** *fastforce*
moreover have $(C,s1)\#(C, t1) \# xsa = map\ (lift\text{-}catch\ Q)\ ((P,s1)\#(P,t1)\#xsa')@((Skip,snd(last\ ((P,s1)\#(P,t1)\#xsa')))\#ys)$
using *xs C* **unfolding** *lift-catch-def*
by *auto*
ultimately show *?case* **by** *fastforce*
next
case (*CptnModNestSkip*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Catch fst-conv nth-Cons-0*)
next
case (*CptnModNestThrow*)
thus *?case* **by** (*metis SmallStepCon.redex-not-Catch fst-conv nth-Cons-0*)
next
case (*CptnModNestCatch1 n* Γ *P0 sa xsa zs P1*)
thus *?case*
proof −
have $\forall c\ x.$ (*LanguageCon.com.Catch c P1*, x) # *zs* = *map* (*lift-catch P1*) ((c, x) # *xsa*)
using *Cons-lift-catch CptnModNestCatch1.hyps*(*3*) **by** *blast*
then have (*P0*, *sa*) # *xsa* = []
by (*metis* (*no-types*) *CptnModNestCatch1.prems*(*3*) *LanguageCon.com.distinct*(*19*) *One-nat-def last-conv-nth last-lift-catch map-is-Nil-conv*)
then show *?thesis*
by *force*
qed
next
case (*CptnModNestCatch2 n* Γ *P1 sa xsa ys zs Q1*)
then have $P1 = P \wedge Q1 = Q \wedge sa = s$ **by** *auto*
moreover then have $(\Gamma, (P1,sa) \# xsa)\in$ *cpn n* Γ *P s*
using *CptnModNestCatch2*(*1*)
by (*simp add*: *cpn-def cptn-eq-cptn-mod-set*)
moreover obtain s' **where** *last zs*=(*Skip*, s')
proof −
assume *a1*: $\bigwedge s'$. *last zs* = (*LanguageCon.com.Skip*, s') $\Longrightarrow$ *thesis*
have $\exists x$. *last zs* = (*LanguageCon.com.Skip*, x)
by (*metis* (*no-types*) *CptnModNestCatch2.hyps*(*6*) *CptnModNestCatch2.prems*(*3*) *append-is-Nil-conv last-ConsR list.simps*(*3*) *prod.exhaust-sel*)
then show *?thesis*
using *a1* **by** *metis*
qed
ultimately show *?case*

using *Cons-lift-catch-append CptnModNestCatch2.hyps*(*6*) **by** *fastforce*
next
case (*CptnModNestCatch3 n* Γ *P0 sa xsa sa′ P1 ys zs*)
then have *P0* = *P* ∧ *P1* = *Q* ∧ *s*=*Normal sa* **by** *auto*
then obtain *i* **where** *zs*:*fst* (*zs*!*i*) = *Q* ∧ (*i*< (*length zs*))
using *CptnModNestCatch3*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ fst-conv length-Cons length-append nth-append-length zero-less-Suc zero-less-diff*)
then have *si*:*Suc i*< *length* ((*Catch P0 P1*,*Normal sa*)#*zs*) **by** *fastforce*
then have *fst* (((*Seq P0 P1*, *Normal sa*) # *zs*)!*Suc i*) = *Q* **using** *zs* **by** *fastforce*

thus *?case* **using** *CptnModNestCatch3*(*9*) *zs*
by (*metis si nth-Cons-Suc*)
qed (*auto*)

lemma *Catch-sound3*:
assumes
a0:(Γ,*x*)∈*cptn* **and**
a1:*x*!*0* = ((*Catch P Q*),*s*) **and**
a2:∀ *i*<*length x*. *fst* (*x*!*i*)≠ *Q* **and**
a3:*fst*(*last x*) = *Throw* **and**
a4:*env-tran-right* Γ *x rely*
shows
False
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *P s*)
case (*CptnOne* Γ *C s1*) **thus** *?case* **by** *auto*
next
case (*CptnEnv* Γ *C st t xsa*)
thus *?case*
proof −
have *f1*: *env-tran-right* Γ ((*C*, *t*) # *xsa*) *rely*
using *CptnEnv.prems*(*4*) *env-tran-tail* **by** *blast*
have *LanguageCon.com.Catch P Q* = *C*
using *CptnEnv.prems*(*1*) **by** *auto*
then show *?thesis*
using *f1 CptnEnv.hyps*(*3*) *CptnEnv.prems*(*2*) *CptnEnv.prems*(*3*) **by** *moura*
qed
next
case (*CptnComp* Γ *C st C′ st′ xsa*)
then have *c-catch*:*C* = (*Catch P Q*) ∧ *st* = *s* **by** *force*
from *CptnComp* **show** *?case* **proof**(*cases*)
case (*Catchc P1 P1′ P2*) **thus** *?thesis*
proof −
have *f1*: *env-tran-right* Γ ((*C′*, *st′*) # *xsa*) *rely*
using *CptnComp.prems*(*4*) *env-tran-tail* **by** *blast*
have *Q* = *P2*
using *c-catch Catchc*(*1*) **by** *blast*
then show *?thesis*

using *f1 CptnComp.hyps*(*3*) *CptnComp.prems*(*2*) *CptnComp.prems*(*3*) *Catchc*(*2*) **by** *moura*
qed
next
case (*CatchSkipc*) **thus** *?thesis*
proof −
have *fst* (((*C′*, *st′*) # *xsa*) ! *0*) = *LanguageCon.com.Skip*
by (*simp add*: *local.CatchSkipc*(*2*))
then show *?thesis*
by (*metis* (*no-types*) *CptnComp.hyps*(*2*) *CptnComp.prems*(*3*) *LanguageCon.com.distinct*(*17*)
last-ConsR last-length length-Cons lessI list.simps(*3*) *zero-skip-all-skip*)
qed
next
case (*SeqThrowc C2 s′*) **thus** *?thesis*
by (*simp add*: *c-catch*)
next
case (*FaultPropc*) **thus** *?thesis*
using *c-catch redex-not-Catch* **by** *blast*
next
case (*StuckPropc*) **thus** *?thesis*
using *c-catch redex-not-Catch* **by** *blast*
next
case (*AbruptPropc*) **thus** *?thesis*
using *c-catch redex-not-Catch* **by** *blast*
qed (*auto*)
qed

lemma *catch-map-xs-ys′*:
assumes
a0:(*n*, Γ, (*P0*, *sa*) # *xsa*) ∈ *cptn-mod-nest-call* **and**
a1:*fst* (*last* ((*P0*, *sa*) # *xsa*)) = *C* **and**
a2:(*n*,Γ, (*P1*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call* **and**
a3:*zs* = *map* (*lift-catch P1*) *xsa* @ (*P1*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys* **and**
a4:((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! *0* = (*LanguageCon.com.Catch P Q*, *s*) **and**
a5:*i* < *length* ((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ∧ ((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! *i* = (*Q*, *sj*) **and**
a6:∀ *j*<*i*. *fst* (((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! *j*) ≠ *Q*
shows
∃ *xs ys*. (Γ, *xs*) ∈ *cpn n* Γ *P s* ∧
(Γ, *ys*) ∈ *cpn n* Γ *Q* (*snd* (*xs* ! (*i* − *1*))) ∧ (*LanguageCon.com.Catch P0 P1*, *sa*) # *zs* = *map* (*lift-catch Q*) *xs* @ *ys*
proof −
let *?P0* = (*P0*, *sa*) # *xsa*
have *P-Q*:*P*=*P0* ∧ *s*=*sa* ∧ *Q* = *P1* **using** *a4* **by** *force*
have *i*:*i*=(*length* ((*P0*, *sa*) # *xsa*))
proof (*cases i*=(*length* ((*P0*, *sa*) # *xsa*)))
case *True* **thus** *?thesis* **by** *auto*

next
case *False*
then have *i*:*i*<(*length* ((*P0*, *sa*) # *xsa*)) ∨ *i* > (*length* ((*P0*, *sa*) # *xsa*)) **by** *auto*
{
assume *i*:*i*<(*length* ((*P0*, *sa*) # *xsa*))
then have *eq-map*:((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! *i* = *map* (*lift-catch P1*) ((*P0*, *sa*) # *xsa*) ! *i*
using *a3 Cons-lift-catch-append*
by (*metis* (*no-types*, *lifting*) *length-map nth-append*)
then have ∃ *ci si*. *map* (*lift-catch P1*) ((*P0*, *sa*) # *xsa*) ! *i* = (*Catch ci P1*,*si*)
using *i* **unfolding** *lift-catch-def*
by (*metis a5 eq-map fst-conv length-map map-lift-catch-all-catch*)
then have ((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! *i* ≠ (*Q*, *sj*)
using *P-Q eq-map* **by** *fastforce*
then have *?thesis* **using** *a5* **by** *auto*
}note *l*=*this*
{
assume *i*:*i*>(*length* ((*P0*, *sa*) # *xsa*))
have *fst* (((*LanguageCon.com.Catch P0 P1*, *sa*) # *zs*) ! (*length ?P0*)) = *Q*
using *a3 P-Q Cons-lift-catch-append* **by** (*metis fstI length-map nth-append-length*)

then have *?thesis* **using** *a6 i* **by** *auto*
}
thus *?thesis* **using** *l i* **by** *auto*
qed
then have (Γ, (*P0*, *sa*) # *xsa*) ∈ *cpn n* Γ *P s*
using *a0 P-Q* **unfolding** *cpn-def* **by** *fastforce*
also have (Γ, (*P1*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ *cpn n* Γ *Q* (*snd* (*?P0* ! ((*length ?P0*) −*1*)))
using *a3 cptn-eq-cptn-mod P-Q* **unfolding** *cpn-def*
proof −
have (*n*, Γ, (*Q*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ *cptn-mod-nest-call*
using *a2 P-Q* **by** *blast*
then have (Γ, (*Q*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ {(*f*, *ps*). *ps* ! *0* = (*Q*, *snd* (((*P0*, *sa*) # *xsa*) ! (*Suc* (*length xsa*) − *1*))) ∧
(*n*, Γ, *ps*) ∈ *cptn-mod-nest-call* ∧ *f* = Γ}
by (*simp add*: *cptn-eq-cptn-mod last-length*)
then show (Γ, (*P1*, *snd* (*last* ((*P0*, *sa*) # *xsa*))) # *ys*) ∈ {(*f*, *ps*). *ps* ! *0* = (*Q*, *snd* (((*P0*, *sa*) # *xsa*) ! (*length* ((*P0*, *sa*) # *xsa*) − *1*))) ∧ (*n*,Γ, *ps*) ∈ *cptn-mod-nest-call* ∧ *f* = Γ}
using *P-Q* **by** *force*
qed
ultimately show *?thesis* **using** *a3 P-Q i* **using** *Cons-lift-catch-append* **by** *blast*
qed

lemma *Catch-sound4*:

assumes
a0:(n,Γ,x)∈*cptn-mod-nest-call* **and**
a1:x!*0* = ((*Catch P Q*),s) **and**
a2:i<*length* x ∧ x!i=(Q,*sj*) **and**
a3:∀ j<i. *fst*(x!j)≠Q **and**
a4:*env-tran-right* Γ x *rely*
shows
∃ *xs ys*. (Γ,*xs*) ∈ (*cpn n* Γ *P s*) ∧ (Γ,*ys*) ∈ (*cpn n* Γ *Q* (*snd* (*xs*!(i−*1*)))) ∧ x = (*map* (*lift-catch Q*) *xs*)@*ys*
using *a0 a1 a2 a3 a4*
proof (*induct arbitrary*: *i sj P s*)
case (*CptnModNestEnv* Γ *C st t n xsa*)
have *a1*:*Catch P Q* ≠ *Q* **by** *simp*
then have *C-catch*:*C*=(*Catch P Q*) **using** *CptnModNestEnv* **by** *fastforce*
then have *fst*((((*C*, *st*) # (*C*, *t*) # *xsa*)!*0*) ≠*Q* **using** *CptnEnv a1* **by** *auto*
moreover have *fst*((((*C*, *st*) # (*C*, *t*) # *xsa*)!*1*) ≠*Q* **using** *CptnModNestEnv a1* **by** *auto*
moreover have *fst*((((*C*, *st*) # (*C*, *t*) # *xsa*)!*i*) =*Q* **using** *CptnModNestEnv* **by** *auto*
ultimately have *i-suc*: *i*> (*Suc 0*) **using** *CptnModNestEnv*
by (*metis Suc-eq-plus1 Suc-lessI add.left-neutral neq0-conv*)
then obtain *i′* **where** *i′*:*i*=*Suc i′* **by** (*meson lessE*)
then have *i-minus*:*i′*=*i*−*1* **by** *auto*
have ((*C*, *t*) # *xsa*) ! *0* = ((*Catch P Q*), *t*)
using *CptnModNestEnv* **by** *auto*
moreover have *i′*< *length* ((*C*,*t*)#*xsa*) ∧ ((*C*,*t*)#*xsa*)!*i′* = (*Q*,*sj*)
using *i′ CptnModNestEnv*(*5*) **by** *force*
moreover have ∀ *j*<*i′*. *fst* (((*C*, *t*) # *xsa*) ! *j*) ≠ *Q*
using *i′ CptnModNestEnv*(*6*) **by** *force*
ultimately have *hyp*:∃ *xs ys*.
(Γ, *xs*) ∈ *cpn n* Γ *P t* ∧
(Γ, *ys*) ∈ *cpn n* Γ *Q* (*snd* (*xs* ! (*i′*−*1*))) ∧ (*C*, *t*) # *xsa* = *map* (*lift-catch Q*) *xs* @ *ys*
using *CptnModNestEnv*(*3*) *env-tran-tail CptnModNestEnv.prems*(*4*) **by** *blast*
then obtain *xs ys* **where** *xs-cp*:(Γ, *xs*) ∈ *cpn n* Γ *P t* ∧
(Γ, *ys*) ∈ *cpn n* Γ *Q* (*snd* (*xs* ! (*i′*−*1*))) ∧ (*C*, *t*) # *xsa* = *map* (*lift-catch Q*) *xs* @ *ys*
by *fast*
have (Γ, (*P*, *s*)#*xs*) ∈ *cpn n* Γ *P s*
proof −
have *xs*!*0* = (*P*,*t*)
using *xs-cp* **unfolding** *cpn-def* **by** *blast*
moreover have *xs*≠[]
using *cpn-def CptnEmpty xs-cp* **by** *blast*
ultimately obtain *xs′* **where** *xs′*:(*n*,Γ, (*P*,*t*)#*xs′*) ∈ *cptn-mod-nest-call* ∧ *xs*=(*P*,*t*)#*xs′*
using *SmallStepCon.nth-tl xs-cp* **unfolding** *cpn-def* **by** *force*
thus *?thesis* **using** *cpn-def*
proof −

```
      have (Catch P Q, s) = (C, st)
        using CptnModNestEnv.prems(1) by auto
      then have Γ⊢c (P, s) →e (P, t)
        using Catch-env-P CptnModNestEnv(1) by blast
      then show ?thesis
        by (simp add: cpn-def cptn-mod-nest-call.CptnModNestEnv xs')
    qed
  qed
  thus ?case
   using i-suc Cons-lift-catch-append CptnModNestEnv.prems(1) i' i-minus xs-cp
    by fastforce
next
  case (CptnModNestSkip Γ P s t n xs)
  then show ?case
    by (metis (no-types) CptnModNestSkip.hyps(2) CptnModNestSkip.prems(1)
    fst-conv nth-Cons-0 redex-not-Catch)
next
  case (CptnModNestThrow Γ P s t n xs)
  then show ?case
   by (metis (no-types) CptnModNestThrow.hyps(2) CptnModNestThrow.prems(1)

      fst-conv nth-Cons-0 redex-not-Catch)
next
  case (CptnModNestCatch1 n Γ P0 s xs zs P1)
  then show ?case
    by (metis Catch-not-c Cons-lift-catch LanguageCon.com.inject(9)
          fst-conv map-lift-catch-all-catch nth-Cons-0)
next
  case (CptnModNestCatch2 n Γ P1 sa xsa  ys zs Q1)
  then have P-Q:P=P1 ∧ Q = Q1 by force
  thus ?case
  proof (cases Q1 = Skip)
    case True thus ?thesis using catch-map-xs-ys'
      CptnModNestCatch2 by blast
  next
    case False note q-not-throw=this
    have ∀x. x< length ((LanguageCon.com.Catch P1 Q1, sa) # zs) ⟶
              ((LanguageCon.com.Catch P1 Q1,  sa) # zs) ! x ≠ (Q, sj) using
CptnModNestCatch2
    proof –
    {
      fix x
      assume x-less:x< length ((LanguageCon.com.Catch P1 Q1,  sa) # zs)
      have ((LanguageCon.com.Catch P1 Q1,  sa) # zs) ! x ≠ (Q, sj)
     proof (cases x < length ((LanguageCon.com.Catch P1 Q1, sa)#map (lift-catch
Q1) xsa))
        case True
        then have eq-map:((LanguageCon.com.Catch P1 Q1, sa) # zs) ! x = map
(lift-catch Q1) ((P1, sa) # xsa) ! x
```

by (*metis* (*no-types*) *Cons-lift-catch Cons-lift-catch-append CptnModNest-Catch2*(*6*) *True nth-append*)
then have ∃ *ci si*. *map* (*lift-catch Q1*) ((*P1*, *sa*) # *xsa*) ! *x* = (*Catch ci Q1*,*si*)
using *True* **unfolding** *lift-catch-def*
by (*metis Cons-lift-catch True eq-map map-lift-catch-all-catch surjective-pairing*)
then have ((*LanguageCon.com.Catch P1 Q1*, *sa*) # *zs*) ! *x* ≠ (*Q*, *sj*)
using *P-Q eq-map* **by** *fastforce*
thus *?thesis* **using** *CptnModNestCatch2*(*10*) **by** *auto*
next
case *False*
let *?s'* = *snd* (*last* ((*P1*, *sa*) # *xsa*))
have *all-throw*:∀ *i*<*length* ((*LanguageCon.com.Skip*, *?s'*)# *ys*).
fst (((*Skip*, *?s'*)# *ys*)!*i*) = *Skip* **using** *CptnModNestCatch2*
by (*metis cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod skip-all-skip*)
then have
∀ *x*≥ *length* ((*LanguageCon.com.Catch P1 Q1*, *sa*) # *map* (*lift-catch Q1*) *xsa*).
x<*length* (((*LanguageCon.com.Catch P1 Q1*, *sa*) # *zs*)) ⟶
fst (((*LanguageCon.com.Catch P1 Q1*, *sa*) # *zs*) ! *x*) = *Skip*
using *Cons-lift-catch Cons-lift-catch-append CptnModNestCatch2.hyps*(*1*) *CptnModNestCatch2.hyps*(*3*)
CptnModNestCatch2.hyps(*4*) *CptnModNestCatch2.hyps*(*6*) *cptn-eq-cptn-mod-set*

cptn-mod-nest-call.CptnModNestCatch2 cptn-mod-nest-cptn-mod skip-com-all-skip
proof−
{
fix *x*
assume *a1*:*x*≥ *length* ((*Catch P1 Q1*, *sa*) # *map* (*lift-catch Q1*) *xsa*) **and**
a2:*x*<*length* (((*Catch P1 Q1*, *sa*) # *zs*))
then have ((*Catch P1 Q1*, *sa*) # *zs*) ! *x* =
((*Skip*, *?s'*)# *ys*) !(*x* − (*length* ((*Catch P1 Q1*, *sa*) # *map* (*lift-catch Q1*) *xsa*)))
using *CptnModNestCatch2*(*6*) **by** (*metis Cons-lift-catch Cons-lift-catch-append not-le nth-append*)
then have*fst* (((*Catch P1 Q1*, *sa*) # *zs*) ! *x*) = *Skip*
using *all-throw a1 a2 CptnModNestCatch2.hyps*(*6*) **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
thus *?thesis* **using** *False q-not-throw P-Q x-less*
by (*metis fst-conv not-le*)
qed
} **thus** *?thesis* **by** *auto*
qed
thus *?thesis* **using** *CptnModNestCatch2*(*8*) **by** *fastforce*
qed
next
case (*CptnModNestCatch3 n* Γ *P1 sa xsa ys zs Q1*)

then show *?case* **using** *catch-map-xs-ys'*[*OF CptnModNestCatch3*(*1*) *CptnModNestCatch3*(*3*) *CptnModNestCatch3*(*5*)
CptnModNestCatch3(*7*) *CptnModNestCatch3*(*8*) *CptnModNestCatch3*(*9*)]
by *blast*
qed(*auto*)

inductive-cases *stepc-elim-cases-Catch-throw*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (Throw,\ Normal\ s1)$

inductive-cases *stepc-elim-cases-Catch-skip-c2*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (c2,s)$

inductive-cases *stepc-elim-cases-Catch-skip-2*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (Skip,\ s)$

lemma *catch-skip-throw*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (c2,s) \implies (c2= Skip \wedge c1=Skip) \vee (c1=Throw \wedge (\exists s2'.\ s=Normal\ s2'))$
apply (*rule stepc-elim-cases-Catch-skip-c2*)
apply *fastforce*
apply (*auto*)+
using *redex-not-Catch* **apply** *auto*
done

lemma *catch-skip-throw1*:
$\Gamma\vdash_c(Catch\ c1\ c2,s) \rightarrow (Skip,s) \implies (c1=Skip) \vee (c1=Throw \wedge (\exists s2'.\ s=Normal\ s2') \wedge c2 = Skip)$
apply (*rule stepc-elim-cases-Catch-skip-2*)
using *redex-not-Catch* **apply** *auto*
using *redex-not-Catch* **by** *auto*

lemma *Catch-sound*:
$\Gamma,\Theta \vdash_{/F} c1\ sat\ [p,\ R,\ G,\ q,r] \implies$
$\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p,\ R,\ G,\ q,r] \implies$
$\Gamma,\Theta \vdash_{/F} c2\ sat\ [r,\ R,\ G,\ q,a] \implies$
$\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [r,\ R,\ G,\ q,a] \implies$
$Sta\ q\ R \implies (\forall s.\ (Normal\ s,Normal\ s) \in G) \implies$
$\Gamma,\Theta \models n_{/F}\ (Catch\ c1\ c2)\ sat\ [p,\ R,\ G,\ q,a]$
proof −
assume
a0:$\Gamma,\Theta \vdash_{/F} c1\ sat\ [p,\ R,\ G,\ q,r]$ **and**
a1:$\forall n.\ \Gamma,\Theta \models n_{/F}\ c1\ sat\ [p,\ R,\ G,\ q,r]$ **and**
a2:$\Gamma,\Theta \vdash_{/F} c2\ sat\ [r,\ R,\ G,\ q,a]$ **and**
a3: $\forall n.\ \Gamma,\Theta \models n_{/F}\ c2\ sat\ [r,\ R,\ G,\ q,a]$ **and**
a4: *Sta q R* **and**
a5: $(\forall s.\ (Normal\ s,\ Normal\ s) \in G)$

{
  fix *s*
  assume *all-call*:$\forall$ (*c*,*p*,*R*,*G*,*q*,*a*)$\in$ Θ. Γ $\models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]
  then have *a1*:Γ $\models n_{/F}$ *c1 sat* [*p*, *R*, *G*, *q*,*r*]
    using *a1 com-cvalidityn-def* **by** *fastforce*
  then have *a3*: Γ $\models n_{/F}$ *c2 sat* [*r*, *R*, *G*, *q*,*a*]
    using *a3 com-cvalidityn-def all-call* **by** *fastforce*
  have *cpn n* Γ (*Catch c1 c2*) *s* $\cap$ *assum*(*p*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
  proof –
  {
    fix *c*
    assume *a10*:*c* $\in$ *cpn n* Γ (*Catch c1 c2*) *s* **and** *a11*:*c* $\in$ *assum*(*p*, *R*)
    then have *a10'*:*c* $\in$ *cp* Γ (*Catch c1 c2*) *s*
      unfolding *cpn-def cp-def*
      using *cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *blast*
    obtain Γ*1 l* **where** *c-prod*:*c*=(Γ*1*,*l*) **by** *fastforce*
    have *cp*:*l*!*0*=((*Catch c1 c2*),*s*) $\wedge$ (*n*,Γ,*l*) $\in$ *cptn-mod-nest-call* $\wedge$ Γ=Γ*1*
      using *a10 cpn-def c-prod* **by** *fastforce*
    then have *cp'*:*l*!*0*=((*Catch c1 c2*),*s*) $\wedge$ (Γ,*l*) $\in$ *cptn* $\wedge$ Γ=Γ*1*
      using *cptn-eq-cptn-mod-nest* **by** *auto*
    have Γ*1*:(Γ, *l*) = *c* **using** *c-prod cp* **by** *blast*
    have *c* $\in$ *comm*(*G*, (*q*,*a*)) *F*
    proof –
    {
     assume *l-f*:*snd* (*last l*) $\notin$ *Fault* ‘ *F*
     have *assum*:*snd*(*l*!*0*) $\in$ *Normal* ‘ (*p*) $\wedge$ ($\forall$ *i*. *Suc i*<*length l* $\longrightarrow$
              (Γ*1*)$\vdash_c$(*l*!*i*) $\rightarrow_e$ (*l*!(*Suc i*)) $\longrightarrow$
              (*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) $\in$ *R*)
     using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
     then have *env-tran*:*env-tran* Γ *p l R* **using** *env-tran-def cp* **by** *blast*
     then have *env-tran-right*: *env-tran-right* Γ *l R*
       using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
     have ($\forall$ *i*. *Suc i*<*length l* $\longrightarrow$
            $\Gamma$$\vdash_c$(*l*!*i*) $\rightarrow$ (*l*!(*Suc i*)) $\longrightarrow$
              (*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) $\in$ *G*)$\wedge$
           (*final* (*last l*) $\longrightarrow$
                ((*fst* (*last l*) = *Skip* $\wedge$
                 snd (*last l*) $\in$ *Normal* ‘ *q*)) $\vee$
                 (*fst* (*last l*) = *Throw* $\wedge$
                 snd (*last l*) $\in$ *Normal* ‘ (*a*)))
     proof (*cases* $\forall$ *i*<*length l*. *fst* (*l*!*i*)$\neq$ *c2*)
       case *True*
       then have *no-c2*:$\forall$ *i*<*length l*. *fst* (*l*!*i*)$\neq$ *c2* **by** *assumption*
       show *?thesis*
       proof (*cases final* (*last l*))
         case *True*
         then obtain *s'* **where** *fst* (*last l*) = *Skip* $\vee$ (*fst* (*last l*) = *Throw* $\wedge$ *snd* (*last l*) = *Normal s'*)
           using *final-def* **by** *fast*

thus *?thesis*
proof
assume *fst* (*last l*)= *LanguageCon.com.Throw* ∧ *snd* (*last l*) = *Normal* *s'*
then have *False* **using** *no-c2 env-tran-right cp' cptn-eq-cptn-mod-set Catch-sound3*
by *blast*
thus *?thesis* **by** *auto*
next
assume *asm0*:*fst* (*last l*) = *Skip*
then obtain *lc1 ys* **where** *cp-lc1*:(Γ,*lc1*) ∈ *cpn n* Γ *c1 s* ∧ *l* = ((*map* (*lift-catch c2*) *lc1*)@((*Skip*,*snd*(*last lc1*))#*ys*))
using *Catch-sound2 cp cptn-eq-cptn-mod-set env-tran-right no-c2* **by** *blast*
then have *cp-lc1'*:(Γ,*lc1*) ∈ *cp* Γ *c1 s*
unfolding *cpn-def cp-def*
using *cptn-if-cptn-mod cptn-mod-nest-cptn-mod* **by** *fastforce*
let *?m-lc1* = *map* (*lift-catch c2*) *lc1*
let *?lm-lc1* = (*length ?m-lc1*)
let *?last-m-lc1* = *?m-lc1*!(*?lm-lc1*−*1*)
have *lc1-not-empty*:*lc1* ≠ []
using Γ*1 a10' cpn-def cp-lc1 cp* **by** *auto*
then have *map-cp*:(Γ,*?m-lc1*) ∈ *cpn n* Γ (*Catch c1 c2*) *s*
proof −
have *f1*: *lc1* ! *0* = (*c1*, *s*) ∧ (*n*, Γ, *lc1*) ∈ *cptn-mod-nest-call* ∧ Γ = Γ
using *cp-lc1* **unfolding** *cpn-def* **by** *blast*
then have *f2*: (*n*,Γ, *?m-lc1*) ∈ *cptn-mod-nest-call* **using** *lc1-not-empty*
by (*metis Cons-lift-catch SmallStepCon.nth-tl cptn-mod-nest-call.CptnModNestCatch1*)
then show *?thesis*
using *f2 f1 lc1-not-empty* **by** (*simp add*: *cpn-def lift-catch-def*)
qed
then have *map-cp'*:(Γ,*?m-lc1*) ∈ *cp* Γ (*Catch c1 c2*) *s*
unfolding *cp-def cpn-def*
using *cp-def cp-lc1' lift-catch-is-cptn* **by** *fastforce*
also have *map-assum*:(Γ,*?m-lc1*) ∈ *assum* (*p*,*R*)
using *sub-assum a10 a11* Γ*1 cp-lc1 lc1-not-empty*
by (*metis SmallStepCon.nth-tl map-is-Nil-conv*)
ultimately have ((Γ,*lc1*) ∈ *assum*(*p*, *R*))
using Γ*1 assum-map-catch cp-lc1'* **by** *blast*
then have *lc1-comm*:(Γ,*lc1*) ∈ *comm*(*G*, (*q*,*r*)) *F*
using *a1 cp-lc1* **by** (*meson IntI com-validityn-def contra-subsetD*)
then have *m-lc1-comm*:(Γ,*?m-lc1*) ∈ *comm*(*G*, (*q*,*r*)) *F*
using *map-cp map-assum comm-map-catch cp-lc1' map-cp'* **by** *blast*
then have *last-m-lc1*:*last* (*?m-lc1*) = (*Catch* (*fst* (*last lc1*)) *c2*,*snd* (*last lc1*))
proof −
have *a000*:∀ *p c*. (*LanguageCon.com.Catch* (*fst p*) *c*, *snd p*) = *lift-catch* *c p*
using *Cons-lift-catch* **by** *force*

then show *?thesis*
by (*simp add*: *last-map a000 lc1-not-empty*)
qed
then have *last-length*:*last* (*?m-lc1*) = *?last-m-lc1*
using *lc1-not-empty last-conv-nth list.map-disc-iff* **by** *blast*
then have *l-map*:*l*!(*?lm-lc1−1*)= *?last-m-lc1*
using *cp-lc1*
by (*simp add*:*lc1-not-empty nth-append*)
then have *lm-lc1*:*l*!(*?lm-lc1*) = (*Skip*, *snd* (*last lc1*))
using *cp-lc1* **by** (*meson nth-append-length*)
then have *step*:$\Gamma\vdash_c$(*l*!(*?lm-lc1−1*)) → (*l*!(*?lm-lc1*))
proof −
have $\Gamma\vdash_c$(*l*!(*?lm-lc1−1*)) $\rightarrow_{ce}$ (*l*!(*?lm-lc1*))
proof −
have *f1*: *0* ≤ *length ys*
by *blast*
moreover have *Suc* (*length* (*map* (*lift-catch c2*) *lc1*) + *length ys*) =
length (*map* (*lift-catch c2*) *lc1* @ (*LanguageCon.com.Skip*, *snd* (*last lc1*)) # *ys*)
by *force*
ultimately show *?thesis*
by (*metis* (*no-types*) *Suc-diff-1 Suc-eq-plus1 cp′ cp-lc1 cptn-tran-ce-i*
lc1-not-empty le-add-same-cancel1 length-greater-0-conv
less-Suc-eq-le list.map-disc-iff)
qed
moreover have ¬$\Gamma\vdash_c$(*l*!(*?lm-lc1−1*)) $\rightarrow_e$ (*l*!(*?lm-lc1*))
using *last-m-lc1 last-length l-map*
proof −
have (*LanguageCon.com.Catch* (*fst* (*last lc1*)) *c2*, *snd* (*last lc1*)) = *l* ! (*length* (*map* (*lift-catch c2*) *lc1*) − *1*)
using *l-map last-m-lc1 local.last-length* **by** *presburger*
then show *?thesis*
by (*metis LanguageCon.com.simps*(*30*) *env-c-c′ lm-lc1*)
qed
ultimately show *?thesis* **using** *step-ce-elim-cases* **by** *blast*
qed
have *last-lc1-suc*:*snd* (*l*!(*?lm-lc1−1*)) = *snd* (*l*!*?lm-lc1*)
using *l-map last-m-lc1 lm-lc1 local.last-length* **by** *force*
then have *step-catch*:$\Gamma\vdash_c$(*Catch* (*fst* (*last lc1*)) *c2*,*snd* (*last lc1*)) → (*Skip*, *snd* (*last lc1*))
using *l-map last-m-lc1 lm-lc1 local.last-length local.step*
by *presburger*
then obtain *s2′* **where**
last-lc1:*fst* (*last lc1*) = *Skip* ∨
fst (*last lc1*) = *Throw* ∧ (*snd* (*last lc1*) = *Normal s2′*) ∧ *c2* = *Skip*
using *catch-skip-throw1* **by** *fastforce*
then have *last-lc1-skip*:*fst* (*last lc1*) = *Skip*

proof
  assume *fst* (*last lc1*) = *LanguageCon.com.Throw* ∧
      snd (*last lc1*) = *Normal s2′* ∧ *c2* = *LanguageCon.com.Skip*
  thus *?thesis* **using** *no-c2 asm0*
    by (*simp add*: *cp-lc1 last-conv-nth*)
qed *auto*
have *last-not-F*:*snd* (*last ?m-lc1*) ∉ *Fault* ‘ *F*
proof −
  have *snd ?last-m-lc1* = *snd* (*l*!(*?lm-lc1*−*1*))
    using *l-map* **by** *auto*
  have (*?lm-lc1*−*1*)< *length l***using** *cp-lc1* **by** *fastforce*
  also then have *snd* (*l*!(*?lm-lc1*−*1*))∉ *Fault* ‘ *F*
    using *cp′ cp-lc1 l-f last-not-F*[*of* Γ *l F*]
    by *fastforce*
  ultimately show *?thesis* **using** *l-map last-length* **by** *fastforce*
qed
then have *q-normal*:*snd* (*l*!*?lm-lc1*) ∈ *Normal* ‘ *q*
proof −
  have *last-lc1*:*fst* (*last lc1*) = *Skip*
  using *last-lc1-skip* **by** *fastforce*
  have *final* (*last lc1*) **using** *last-lc1 final-def*
    by *blast*
  then show *?thesis*
    using *lc1-comm last-lc1 last-not-F*
    unfolding *comm-def*
    using *last-lc1-suc comm-dest2 l-map lm-lc1 local.last-length*
    by *force*
qed
then obtain *s1′* **where** *normal-lm-lc1*:*snd* (*l*!*?lm-lc1*) = *Normal s1′*
∧ *s1′* ∈*q*
  by *auto*
have *concl*:(∀ *i ns ns′*. *Suc i*<*length l* ⟶
  Γ⊢$_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
    (*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)
proof−
{ fix *k ns ns′*
  assume *a00*:*Suc k*<*length l* **and**
   a21:Γ⊢$_c$(*l*!*k*) → (*l*!(*Suc k*))
  then have *i-m-l*:∀ *i* <*?lm-lc1* . *l*!*i* = *?m-lc1*!*i*
    using *cp-lc1*
  proof −
    have *map* (*lift c2*) *lc1* ≠ []
      by (*meson lc1-not-empty list.map-disc-iff*)
    then show *?thesis*
      by (*metis* (*no-types*) *cp-lc1 nth-append*)
  qed
  have (*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) ∈ *G*
  proof (*cases Suc k*< *?lm-lc1*)
    case *True*

```
  then have a11': Γ⊢c(?m-lc1!k) → (?m-lc1!(Suc k))
    using a11 i-m-l True
  proof -
    have ∀ n na. ¬ 0 < n − Suc na ∨ na < n
      using diff-Suc-eq-diff-pred zero-less-diff by presburger
    then show ?thesis
      by (metis (no-types) True a21 i-m-l zero-less-diff)
  qed
  then have (snd(?m-lc1!k), snd(?m-lc1!(Suc k))) ∈ G
 using a11' m-lc1-comm True comm-dest1 l-f last-not-F by fastforce
  thus ?thesis using i-m-l True by auto
next
  case False
  then have (Suc k=?lm-lc1) ∨ (Suc k>?lm-lc1) by auto
  thus ?thesis
  proof
    {assume suck:(Suc k=?lm-lc1)
     then have k:k=?lm-lc1−1 by auto
    then obtain s1' where s1'-normal:snd(l!?lm-lc1) = Normal s1'
       using q-normal by fastforce
     have G-s1':(Normal s1', Normal s1')∈ G using a5 by auto
     then show (snd (l!k), snd (l!Suc k)) ∈ G
     proof -
       have snd (l ! k) = Normal s1'
         using k last-lc1-suc s1'-normal by presburger
       then show ?thesis
         using G-s1' s1'-normal suck by force
     qed
    }
  next
  {
    assume a001:Suc k>?lm-lc1
    have ∀ i. i≥(length lc1) ∧ (Suc i < length l) ⟶
          ¬(Γ⊢c(l!i) → (l!(Suc i)))
    using lm-lc1 lc1-not-empty
    proof -
      have env-tran-right Γ1 l R
        by (metis cp env-tran-right)
      then show ?thesis
        using  cp' fst-conv length-map lm-lc1 a001 a21 a00 a4
              normal-lm-lc1
        by (metis (no-types) only-one-component-tran-j)
    qed
    then have ¬(Γ⊢c(l!k) → (l!(Suc k)))
      using a00 a001 by auto
    then show ?thesis using a21 by fastforce
  }
  qed
qed
```

} **thus** *?thesis* **by** *auto*
qed
have *concr*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ' *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ' (*a*)))
proof −
have *l-t*:*fst* (*last l*) = *Skip*
using *lm-lc1* **by** (*simp add*: *asm0*)
have *?lm-lc1* $\leq$ *length l* −*1* **using** *cp-lc1* **by** *fastforce*
also have $\forall$ *i*. *?lm-lc1* $\leq$*i* $\wedge$ *i*<(*length l* −*1*) $\longrightarrow$ $\Gamma\vdash_c$(*l*!*i*) $\rightarrow_e$ (*l*!(*Suc i*))
using *cp′ fst-conv length-map lm-lc1 a4*
normal-lm-lc1 only-one-component-tran-j[*of* Γ *l ?lm-lc1 s1′ q*]
by (*metis Suc-eq-plus1 cptn-tran-ce-i env-tran-right less-diff-conv step-ce-elim-cases*)
ultimately have *snd* (*l* ! (*length l* − *1*)) $\in$ *Normal* ' *q*
using *cp-lc1 q-normal a4 env-tran-right stability*[*of q R l ?lm-lc1* (*length l*) −*1* - Γ]
by *fastforce*
thus *?thesis* **using** *l-t*
by (*simp add*: *cp-lc1 last-conv-nth*)
qed
note *res* = *conjI* [*OF concl concr*]
then show *?thesis* **using** Γ*1 c-prod* **unfolding** *comm-def* **by** *auto*
qed
next
case *False*
then obtain *lc1* **where** *cp-lc1*:(Γ,*lc1*) $\in$ *cpn n* Γ *c1 s* $\wedge$ *l* = *map* (*lift-catch c2*) *lc1*
using *Catch-sound1 False no-c2 env-tran-right cp cptn-eq-cptn-mod-set*
by *blast*
then have *cp-lc1′*:(Γ,*lc1*) $\in$ *cp* Γ *c1 s*
unfolding *cpn-def cp-def*
using *cptn-eq-cptn-mod-nest* **by** *fastforce*
then have ((Γ,*lc1*) $\in$ *assum*(*p*, *R*))
using Γ*1 a10′ a11 assum-map-catch cp-lc1*
by *blast*
then have (Γ, *lc1*)$\in$ *comm*(*G*, (*q*,*r*)) *F* **using** *cp-lc1 a1*
by (*meson IntI com-validityn-def contra-subsetD*)
then have (Γ, *l*)$\in$ *comm*(*G*, (*q*,*r*)) *F*
using *comm-map-catch a10′* Γ*1 cp-lc1 cp-lc1′* **by** *fastforce*
then show *?thesis* **using** *l-f False*
unfolding *comm-def* **by** *fastforce*
qed
next
case *False*
then obtain *k* **where** *k-len*:*k*<*length l* $\wedge$ *fst* (*l* ! *k*) = *c2*

by *blast*
then have $\exists m.\ (m < length\ l \wedge fst\ (l\ !\ m) = c2) \wedge$
$(\forall i{<}m.\ \neg\ (i < length\ l \wedge fst\ (l\ !\ i) = c2))$
using *a0 exists-first-occ*[*of* ($\lambda i.\ i{<}length\ l\ \wedge fst\ (l\ !\ i) = c2$) *k*]
by *blast*
then obtain *i* **where** *a0*:$i{<}length\ l \wedge fst\ (l\ !i) = c2 \wedge$
$(\forall j{<}i.\ (fst\ (l\ !\ j) \neq c2))$
by *fastforce*
then obtain *s2* **where** *li*:$l!i = (c2,s2)$ **by** (*meson eq-fst-iff*)
then obtain *lc1 lc2* **where** *cp-lc1*:$(\Gamma,lc1) \in (cpn\ n\ \Gamma\ c1\ s) \wedge$
$(\Gamma,lc2) \in (cpn\ n\ \Gamma\ c2\ (snd\ (lc1!(i-1)))) \wedge$
$l = (map\ (lift\text{-}catch\ c2)\ lc1)@lc2$
using *Catch-sound4 a0 cp env-tran-right* **by** *blast*
then have *cp-lc1'*:$(\Gamma,lc1) \in (cp\ \Gamma\ c1\ s) \wedge$
$(\Gamma,lc2) \in (cp\ \Gamma\ c2\ (snd\ (lc1!(i-1))))$
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-nest* **by** *fastforce*
have *i-not-fault*:$snd\ (l!i) \notin Fault\ `\ F$ **using** *a0 cp' l-f last-not-F*[*of* Γ *l F*]
by *blast*
have *length-c1-map*:$length\ lc1 = length\ (map\ (lift\text{-}catch\ c2)\ lc1)$
by *fastforce*
then have *i-map*:$i{=}length\ lc1$
using *cp-lc1 li a0* **unfolding** *lift-catch-def*
proof −
assume *a1*: $(\Gamma,\ lc1) \in cpn\ n\ \Gamma\ c1\ s \wedge (\Gamma,\ lc2) \in cpn\ n\ \Gamma\ c2\ (snd\ (lc1\ !\ (i - 1))) \wedge l = map\ (\lambda(P,\ s).\ (Catch\ P\ c2,\ s))\ lc1\ @\ lc2$
have *f2*: $i < length\ l \wedge fst\ (l\ !\ i) = c2 \wedge (\forall n.\ \neg\ n < i \vee fst\ (l\ !\ n) \neq c2)$
using *a0* **by** *blast*
have *f3*: $(Catch\ (fst\ (lc1\ !\ i))\ c2,\ snd\ (lc1\ !\ i)) = lift\text{-}catch\ c2\ (lc1\ !\ i)$
by (*simp add*: *case-prod-unfold lift-catch-def*)
then have $fst\ (l\ !\ length\ lc1) = c2$
using *a1* **by** (*simp add*: *cpn-def nth-append*)
thus *?thesis*
using *f3 f2*
by (*metis* (*no-types*, *lifting*) *Pair-inject a0 cp-lc1 f3*
length-c1-map li linorder-neqE-nat nth-append nth-map seq-and-if-not-eq(*12*))
qed
have *lc2-l*:$\forall j{<}length\ lc2.\ lc2!j{=}l!(i{+}j)$
using *cp-lc1 length-c1-map i-map a0*
by (*metis nth-append-length-plus*)
have *lc1-not-empty*:$lc1 \neq []$
using *cp cp-lc1* **unfolding** *cpn-def* **by** *fastforce*
have *lc2-not-empty*:$lc2 \neq []$
using *cpn-def cp-lc1 a0 i-map* **by** *force*
have *l-is*:$s2{=}\ snd\ (last\ lc1)$
using *cp-lc1 li a0 lc1-not-empty* **unfolding** *cpn-def*
by (*auto simp add*: *i-map last-conv-nth lc2-l*)
let *?m-lc1* = *map* (*lift-catch c2*) *lc1*

have *last-m-lc1*:$l!(i-1) = (Catch\ (fst\ (last\ lc1))\ c2,s2)$
using *i-map cp-lc1 l-is last-lift-catch last-snd-catch lc1-not-empty length-c1-map*
by (*metis* (*no-types, lifting*) *One-nat-def diff-Suc-less last-conv-nth length-greater-0-conv nth-append prod.collapse*)
have *last-mcl1-not-F*:$snd\ (last\ ?m\text{-}lc1) \notin Fault\ `\ F$
by (*metis One-nat-def i-not-fault l-is last-conv-nth last-snd-catch li list.map-disc-iff snd-conv*)
have *map-cp*:$(\Gamma,?m\text{-}lc1) \in cpn\ n\ \Gamma\ (Catch\ c1\ c2)\ s$
proof −
have *f1*: $lc1\ !\ 0 = (c1,\ s) \wedge (n,\Gamma,\ lc1) \in cptn\text{-}mod\text{-}nest\text{-}call \wedge \Gamma = \Gamma$
using *cp-lc1 cpn-def* **by** *blast*
then have *f2*: $(n,\ \Gamma,\ ?m\text{-}lc1) \in cptn\text{-}mod\text{-}nest\text{-}call$ **using** *lc1-not-empty*
by (*metis Cons-lift-catch SmallStepCon.nth-tl cptn-mod-nest-call.CptnModNestCatch1*)

then show *?thesis*
using *f2 f1 lc1-not-empty* **by** (*simp add*: *cpn-def lift-catch-def*)
qed
then have *map-cp'*:$(\Gamma,?m\text{-}lc1) \in cp\ \Gamma\ (Catch\ c1\ c2)\ s$
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-nest* **by** *fastforce*
also have *map-assum*:$(\Gamma,?m\text{-}lc1) \in assum\ (p,R)$
using *sub-assum a10 a11* $\Gamma 1$ *cp-lc1 lc1-not-empty*
by (*metis SmallStepCon.nth-tl map-is-Nil-conv*)
ultimately have $((\Gamma,lc1) \in assum(p,\ R))$
using $\Gamma 1$ *assum-map-catch* **using** *assum-map cp-lc1 cp-lc1'* **by** *blast*

then have *lc1-comm*:$(\Gamma,lc1) \in comm(G,\ (q,r))\ F$
using *a1 cp-lc1* **by** (*meson IntI com-validityn-def contra-subsetD*)
then have *m-lc1-comm*:$(\Gamma,?m\text{-}lc1) \in comm(G,\ (q,r))\ F$
using *map-cp' map-assum comm-map-catch cp-lc1 cp-lc1'* **by** *blast*
then have $\Gamma\vdash_c(l!(i-1)) \rightarrow (l!i)$
proof −
have $\Gamma\vdash_c(l!(i-1)) \rightarrow_{ce} (l!(i))$
by (*metis Suc-eq-plus1 Suc-pred' a0 cp' cptn-tran-ce-i i-map lc1-not-empty length-greater-0-conv*)
moreover have $\neg\Gamma\vdash_c(l!(i-1)) \rightarrow_e (l!i)$
using *li last-m-lc1*
by (*metis* (*no-types, lifting*) *env-c-c' seq-and-if-not-eq*(*12*))
ultimately show *?thesis* **using** *step-ce-elim-cases* **by** *blast*
qed
then have *step*:$\Gamma\vdash_c(Catch\ (fst\ (last\ lc1))\ c2,s2) \rightarrow (c2,\ s2)$
using *last-m-lc1 li* **by** *fastforce*
then obtain *s2'* **where**
last-lc1:$(fst\ (last\ lc1) = Skip \wedge c2 = Skip) \vee$
$fst\ (last\ lc1) = Throw \wedge (s2 = Normal\ s2')$
using *catch-skip-throw* **by** *blast*
have *final*:$final\ (last\ lc1)$
using *last-lc1 l-is* **unfolding** *final-def* **by** *auto*
have *normal-last*:$fst\ (last\ lc1) = Skip \wedge snd\ (last\ lc1) \in Normal\ `\ q \vee$

fst (last lc1) = Throw ∧ *snd (last lc1)* ∈ *Normal ‘ r*

proof –

have *snd (last lc1)* ∉ *Fault ‘ F*

using *i-not-fault l-is li* **by** *auto*

then show *?thesis*

using *final comm-dest2 lc1-comm* **by** *blast*

qed

obtain *s2′* **where** *lastlc1-normal:snd (last lc1) = Normal s2′*

using *normal-last* **by** *blast*

then have *Normals2:s2 = Normal s2′* **by** (*simp add: l-is*)

have *Gs2′*: (*Normal s2′*, *Normal s2′*)∈*G* **using** *a5* **by** *auto*

have *concl*:

(∀ *i. Suc i<length l* ⟶

$\Gamma\vdash_c$(*l*!*i*) → (*l*!(*Suc i*)) ⟶

(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*)

proof–

{ **fix** *k ns ns′*

assume *a00:Suc k<length l* **and**

a21:$\Gamma\vdash_c$(*l*!*k*) → (*l*!(*Suc k*))

have *i-m-l*:∀ *j <i . l*!*j = ?m-lc1*!*j*

proof –

have *map (lift c2) lc1* ≠ []

by (*meson lc1-not-empty list.map-disc-iff*)

then show *?thesis*

using *cp-lc1 i-map length-c1-map* **by** (*fastforce simp:nth-append*)

qed

have (*snd*(*l*!*k*), *snd*(*l*!(*Suc k*))) ∈ *G*

proof (*cases Suc k< i*)

case *True*

then have *a11′*: $\Gamma\vdash_c$(*?m-lc1*!*k*) → (*?m-lc1*!(*Suc k*))

using *a11 i-m-l True*

proof –

have ∀ *n na.* ¬ *0 < n − Suc na* ∨ *na < n*

using *diff-Suc-eq-diff-pred zero-less-diff* **by** *presburger*

then show *?thesis* **using** *True a21 i-m-l* **by** *force*

qed

have *Suc k < length ?m-lc1* **using** *True i-map length-c1-map* **by** *metis*

then have (*snd*(*?m-lc1*!*k*), *snd*(*?m-lc1*!(*Suc k*))) ∈ *G*

using *a11′ last-mcl1-not-F m-lc1-comm True i-map length-c1-map comm-dest1*[*of* Γ]

by *blast*

thus *?thesis* **using** *i-m-l True* **by** *auto*

next

case *False*

have (*Suc k=i*) ∨ (*Suc k>i*) **using** *False* **by** *auto*

thus *?thesis*

proof

{ **assume** *suck*:(*Suc k=i*)

then have k:$k=i-1$ **by** *auto*
then show (*snd* ($l!k$), *snd* ($l!Suc\ k$)) $\in G$
using *Gs2′ Normals2 last-m-lc1 li suck* **by** *auto*
}
next
{
assume *a001*:$Suc\ k>i$
then have k:$k\geq i$ **by** *fastforce*
then obtain k' **where** k':$k=i+k'$
using *add.commute le-Suc-ex* **by** *blast*
{assume *skip*:$c2=Skip$
then have $\forall k.\ k\geq i \wedge (Suc\ k < length\ l) \longrightarrow$
$\neg(\Gamma\vdash_c(l!k) \ \rightarrow (l!(Suc\ k)))$
using *Normals2 li lastlc1-normal a21 a001 a00 a4*
a0 skip env-tran-right cp′
by (*metis SmallStepCon.final-def SmallStepCon.no-step-final′*
Suc-lessD skip-com-all-skip)
then have *?thesis* **using** *a21 a001 k a00* **by** *blast*
} **note** *left=this*
{assume $c2\neq Skip$
then have *fst* (*last lc1*) = *Throw*
using *last-m-lc1 last-lc1* **by** *simp*
then have *s2-normal*:$s2 \in Normal$ ‘ r
using *normal-last lastlc1-normal Normals2*
by *fastforce*
have *length-lc2*:*length l=i+length lc2*
using *i-map cp-lc1* **by** *fastforce*
have (Γ,*lc2*) $\in$ *assum* (r,R)
proof −
have *left*:*snd* (*lc2!0*) $\in$ *Normal* ‘ r
using *li lc2-l s2-normal lc2-not-empty* **by** *fastforce*
{
fix j
assume *j-len*:*Suc j<length lc2* **and**
j-step:$\Gamma\vdash_c(lc2!j) \ \rightarrow_e (lc2!(Suc\ j))$
then have *suc-len*:*Suc* (i + j)<*length l* **using** *j-len length-lc2*
by *fastforce*
also then have $\Gamma\vdash_c(l!(i+j)) \ \rightarrow_e (l!\ (Suc\ (i+\ j)))$
using *lc2-l j-step j-len* **by** *fastforce*
ultimately have (*snd*(*lc2!j*), *snd*(*lc2!*(*Suc j*))) $\in R$
using *assum suc-len lc2-l j-len cp* **by** *fastforce*
}
then show *?thesis* **using** *left*
unfolding *assum-def* **by** *fastforce*
qed
also have (Γ,*lc2*) $\in$ *cpn n* Γ *c2 s2*
using *cp-lc1 i-map l-is last-conv-nth lc1-not-empty* **by** *fastforce*

ultimately have *comm-lc2*:(Γ,*lc2*) $\in$ *comm* (G, (q,a)) F

using *a3* **unfolding** *com-validityn-def* **by** *auto*
have *lc2-last-f*:*snd* (*last lc2*)$\notin$ *Fault* ‘ *F*
using *lc2-l lc2-not-empty l-f cp-lc1* **by** *fastforce*
have *suck'*:*Suc k'* < *length lc2*
using *k' a00 length-lc2* **by** *arith*
moreover then have $\Gamma\vdash_c$(*lc2*!*k'*) $\rightarrow$ (*lc2*!(*Suc k'*))
using *k' lc2-l a21* **by** *fastforce*
ultimately have (*snd* (*lc2*! *k'*), *snd* (*lc2* ! *Suc k'*)) $\in$ *G*
using *comm-lc2 lc2-last-f comm-dest1*[*of* Γ *lc2 G q a F k'*]
by *blast*
then have *?thesis* **using** *suck' lc2-l k'* **by** *fastforce*
}
then show *?thesis* **using** *left* **by** *auto*
}
qed
qed
} **thus** *?thesis* **by** *auto*
qed note *left=this*
have *right*:(*final* (*last l*) $\longrightarrow$
((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ‘ (*a*)))
proof −
{ **assume** *final-l*:*final* (*last l*)
have *eq-last-lc2-l*:*last l=last lc2* **by** (*simp add*: *cp-lc1 lc2-not-empty*)
then have *final-lc2*:*final* (*last lc2*) **using** *final-l* **by** *auto*
{
assume *lst-lc1-skip*:*fst* (*last lc1*) = *Skip*
then have *c2-skip*:*c2* = *Skip*
using *step lastlc1-normal LanguageCon.com.distinct*(*17*) *last-lc1*
by *auto*
have *Skip*:*fst* (*l*!(*length l* − *1*)) = *Skip*
using *li Normals2 env-tran-right cp' c2-skip a0*
i-skip-all-skip[*of* Γ *l i* (*length l*) − *1* -]
by *fastforce*
have *s2-a*:*s2* $\in$ *Normal* ‘ *q*
using *normal-last*
by (*simp add*: *lst-lc1-skip l-is*)
then have $\forall$*ia*. *i* $\leq$ *ia* $\wedge$ *ia* < *length l* − *1* $\longrightarrow$ $\Gamma\vdash_c$ *l* ! *ia* $\rightarrow_e$ *l* ! *Suc ia*
using *c2-skip li Normals2 a0 cp' env-tran-right final-def*
by (*metis* (*no-types*, *hide-lams*) *One-nat-def SmallStepCon.no-step-final'*

Suc-lessD add.right-neutral add-Suc-right
cptn-tran-ce-i i-skip-all-skip less-diff-conv step-ce-elim-cases)

then have *snd* (*l*!(*length l* − *1*)) $\in$ *Normal* ‘ *q* $\wedge$ *fst* (*l*!(*length l* − *1*))
= *Skip*
using *a0 s2-a li a4 env-tran-right stability*[*of q R l i* (*length l*) −*1* - Γ]

Skip
by (*metis One-nat-def Suc-pred length-greater-0-conv lessI linorder-not-less list.size*(*3*)
not-less0 not-less-eq-eq snd-conv)
then have ((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ‘ (*a*))
using *a0* **by** (*metis last-conv-nth list.size*(*3*) *not-less0*)
} **note** *left* = *this*
{ **assume** *fst* (*last lc1*) = *Throw*
then have *s2-normal*:*s2* ∈ *Normal* ‘ *r*
using *normal-last lastlc1-normal Normals2*
by *fastforce*
have *length-lc2*:*length l*=*i*+*length lc2*
using *i-map cp-lc1* **by** *fastforce*
have (Γ,*lc2*) ∈ *assum* (*r*,*R*)
proof −
have *left*:*snd* (*lc2*!*0*) ∈ *Normal* ‘ *r*
using *li lc2-l s2-normal lc2-not-empty* **by** *fastforce*
{
fix *j*
assume *j-len*:*Suc j*<*length lc2* **and**
j-step:$\Gamma\vdash_c$(*lc2*!*j*) $\rightarrow_e$ (*lc2*!(*Suc j*))

then have *suc-len*:*Suc* (*i* + *j*)<*length l* **using** *j-len length-lc2*
by *fastforce*
also then have $\Gamma\vdash_c$(*l*!(*i*+*j*)) $\rightarrow_e$ (*l*! (*Suc* (*i*+ *j*)))
using *lc2-l j-step j-len* **by** *fastforce*
ultimately have (*snd*(*lc2*!*j*), *snd*(*lc2*!(*Suc j*))) ∈ *R*
using *assum suc-len lc2-l j-len cp* **by** *fastforce*
}
then show *?thesis* **using** *left*
unfolding *assum-def* **by** *fastforce*
qed
also have (Γ,*lc2*) ∈ *cpn n* Γ *c2 s2*
using *cp-lc1 i-map l-is last-conv-nth lc1-not-empty* **by** *fastforce*
ultimately have *comm-lc2*:(Γ,*lc2*) ∈ *comm* (*G*, (*q*,*a*)) *F*
using *a3* **unfolding** *com-validityn-def* **by** *auto*
have *lc2-last-f*:*snd* (*last lc2*)∉ *Fault* ‘ *F*
using *lc2-l lc2-not-empty l-f cp-lc1* **by** *fastforce*
then have ((*fst* (*last lc2*) = *Skip* ∧
snd (*last lc2*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last lc2*) = *Throw* ∧
snd (*last lc2*) ∈ *Normal* ‘ (*a*))
using *final-lc2 comm-lc2* **unfolding** *comm-def* **by** *auto*
then have ((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ‘ *q*)) ∨
(*fst* (*last l*) = *Throw* ∧

snd (*last l*) $\in$ *Normal* ' (*a*))
using *eq-last-lc2-l* **by** *auto*
}
then have ((*fst* (*last l*) = *Skip* $\wedge$
snd (*last l*) $\in$ *Normal* ' *q*)) $\vee$
(*fst* (*last l*) = *Throw* $\wedge$
snd (*last l*) $\in$ *Normal* ' (*a*))
using *left* **using** *last-lc1* **by** *auto*
} thus *?thesis* **by** *auto* **qed**
thus *?thesis* **using** *left l-f* $\Gamma 1$ **unfolding** *comm-def* **by** *force*
qed
} thus *?thesis* **using** $\Gamma 1$ **unfolding** *comm-def* **by** *auto* **qed**
} thus *?thesis* **by** *auto* **qed**
} thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *DynCom-sound*:
($\forall s \in p$. (($\Gamma,\Theta \vdash_{/F}$ (*c1 s*) *sat* [*p*, *R*, *G*, *q*,*a*]) $\wedge$
($\forall n$. ($\Gamma,\Theta \models n_{/F}$ (*c1 s*) *sat* [*p*,*R*, *G*, *q*,*a*])))) $\Longrightarrow$
($\forall s$. (*Normal s*, *Normal s*) $\in G$) $\Longrightarrow$
(*Sta p R*) $\wedge$ (*Sta q R*) $\wedge$ (*Sta a R*) $\Longrightarrow$
$\Gamma,\Theta \models n_{/F}$ (*DynCom c1*) *sat* [*p*, *R*, *G*, *q*,*a*]
proof –
assume
a0:($\forall s \in p$. (($\Gamma,\Theta \vdash_{/F}$ (*c1 s*) *sat* [*p*, *R*, *G*, *q*,*a*]) $\wedge$
($\forall n$. ($\Gamma,\Theta \models n_{/F}$ (*c1 s*) *sat* [*p*, *R*, *G*, *q*,*a*])))) **and**
a1:$\forall s$. (*Normal s*, *Normal s*) $\in G$ **and**
a2: (*Sta p R*) $\wedge$ (*Sta q R*) $\wedge$ (*Sta a R*)
{
fix *s*
assume *all-DynCom*:$\forall$(*c*,*p*,*R*,*G*,*q*,*a*)$\in \Theta$. $\Gamma \models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]

then have *a0*:($\forall s \in p$. ($\Gamma \models n_{/F}$ (*c1 s*) *sat* [*p*, *R*, *G*, *q*,*a*]))
using *a0* **unfolding** *com-cvalidityn-def* **by** *fastforce*
have *cpn n* Γ(*DynCom c1*) *s* $\cap$ *assum*(*p*, *R*) $\subseteq$ *comm*(*G*, (*q*,*a*)) *F*
proof –
{
fix *c*
assume *a10*:*c* $\in$ *cpn n* Γ (*DynCom c1*) *s* **and** *a11*:*c* $\in$ *assum*(*p*, *R*)
then have *a10'*:*c* $\in$ *cp* Γ (*DynCom c1*) *s*
unfolding *cp-def cpn-def*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
obtain $\Gamma 1$ *l* **where** *c-prod*:*c*=($\Gamma 1$,*l*) **by** *fastforce*
have *c* $\in$ *comm*(*G*, (*q*,*a*)) *F*
proof –
{assume *l-f*:*snd* (*last l*) $\notin$ *Fault* ' *F*

have *cp*:$l!0$=(*DynCom c1*,*s*) ∧ (Γ,*l*) ∈ *cptn* ∧ Γ=Γ*1*
using *a10′ cp-def c-prod* **by** *fastforce*
have Γ*1*:(Γ, *l*) = *c* **using** *c-prod cp* **by** *blast*
have *assum*:*snd*($l!0$) ∈ *Normal* ‘ (*p*) ∧ (∀ *i*. *Suc i*<*length l* ⟶
(Γ*1*)$\vdash_c$($l!i$) $\rightarrow_e$ ($l!$(*Suc i*)) ⟶
(*snd*($l!i$), *snd*($l!$(*Suc i*))) ∈ *R*)
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:*env-tran* Γ *p l R* **using** *env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
obtain *ns* **where** *s-normal*:*s*=*Normal ns* ∧ *ns*∈ *p*
using *cp assum* **by** *fastforce*
have *concl*:(∀ *i*. *Suc i*<*length l* ⟶
Γ*1*$\vdash_c$($l!i$) → ($l!$(*Suc i*)) ⟶
(*snd*($l!i$), *snd*($l!$(*Suc i*))) ∈ *G*)
proof −
{ fix *k ns ns′*
assume *a00*:*Suc k*<*length l* **and**
a21:Γ$\vdash_c$($l!k$) → ($l!$(*Suc k*))
obtain *j* **where** *before-k-all-evnt*:*j*≤*k* ∧ (Γ$\vdash_c$($l!j$) → ($l!$(*Suc j*))) ∧ (∀ *k* < *j*. (Γ$\vdash_c$($l!k$) $\rightarrow_e$ ($l!$(*Suc k*))))
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:(Γ$\vdash_c$(*cj*,*sj*) → (*csj*,*ssj*)) ∧ *cj* = *fst* ($l!j$) ∧ *sj* = *snd* ($l!j$) ∧ *csj* = *fst* ($l!$(*Suc j*)) ∧ *ssj* = *snd*($l!$(*Suc j*))
by *fastforce*
have *k-basic*:*cj* = (*DynCom c1*) ∧ *sj* ∈ *Normal* ‘ (*p*)
using *pair-j before-k-all-evnt a2 cp env-tran-right assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain *s′* **where** *ss*:*sj* = *Normal s′* ∧ *s′*∈ (*p*) **by** *auto*
then have *ssj-normal-s*:*ssj* = *Normal s′*
using *before-k-all-evnt k-basic pair-j a0*
by (*metis snd-conv stepc-Normal-elim-cases*(*10*))
have (*snd*($l!k$), *snd*($l!$(*Suc k*))) ∈ *G*
using *ss a2* **unfolding** *Satis-def*
proof (*cases k*=*j*)
case *True*
have (*Normal s′*, *Normal s′*)∈*G* **using** *a1* **by** *fastforce*
thus (*snd* (*l* ! *k*), *snd* (*l* ! *Suc k*)) ∈ *G*
using *pair-j k-basic True ss ssj-normal-s* **by** *auto*
next
case *False*
have *j-k*:*j*<*k* **using** *before-k-all-evnt False* **by** *fastforce*
thus (*snd* (*l* ! *k*), *snd* (*l* ! *Suc k*)) ∈ *G*
proof −
have *j-length*:*Suc j* < *length l* **using** *a00 before-k-all-evnt* **by** *fastforce*
have *p1*:*s′*∈*p* ∧ *ssj*=*Normal s′* **using** *ss ssj-normal-s* **by** *fastforce*
then have *c1-valid*:(Γ $\models n_{/F}$ (*c1 s′*) *sat* [*p*, *R*, *G*, *q*,*a*])
using *a0* **by** *fastforce*

```
          have cj:csj= (c1 s') using k-basic pair-j ss a0 s-normal
          proof -
            have Γ⊢c (LanguageCon.com.DynCom c1, Normal s') → (csj, ssj)
              using k-basic pair-j ss by force
            then have (csj, ssj) = (c1 s', Normal s')
              by (meson stepc-Normal-elim-cases(10))
            then show ?thesis
              by blast
          qed
         moreover then have cpn n Γ csj ssj ∩ assum(p, R) ⊆ comm(G, (q,a))
F
            using a2 com-validityn-def cj p1 c1-valid by blast
          moreover then have l!(Suc j) = (csj, Normal s')
            using before-k-all-evnt pair-j cj ssj-normal-s
            by fastforce
          ultimately have drop-comm:((Γ, drop (Suc j) l))∈ comm(G, (q,a)) F
            using  p1 j-length  a11 Γ1  ssj-normal-s
           using a10 cpn-assum-induct by fastforce
          then show ?thesis
          using a00 a21  a10' Γ1  j-k j-length l-f
          cptn-comm-induct[of Γ l DynCom c1 s - Suc j G q a F k ]
          unfolding Satis-def by fastforce
        qed
      qed
      } thus ?thesis by (simp add: c-prod cp) qed
      have concr:(final (last l)  ⟶
                ((fst (last l) = Skip ∧
                 snd (last l) ∈ Normal ‘ q)) ∨
                (fst (last l) = Throw ∧
                 snd (last l) ∈ Normal ‘ (a)))
      proof−
      {
        assume valid:final (last l)
        have ∃ k. k≥0 ∧ k<((length l) − 1) ∧ (Γ⊢c(l!k)  → (l!(Suc k))) ∧ final
(l!(Suc k))
        proof −
          have len-l:length l > 0 using cp using cptn.simps by blast
            then obtain a1 l1 where l:l=a1#l1 by (metis SmallStepCon.nth-tl
length-greater-0-conv)
          have last-l:last l = l!(length l−1)
           using last-length [of a1 l1] l by fastforce
          have final-0:¬final(l!0) using cp unfolding final-def by auto
          have 0≤ (length l−1) using len-l last-l by auto
          moreover have (length l−1) < length l using len-l by auto
          moreover have final (l!(length l−1)) using valid last-l by auto
          moreover have fst (l!0) = DynCom c1 using cp by auto
          ultimately show ?thesis
            using a2 cp final-exist-component-tran-final env-tran-right final-0
            by blast
```

qed
then obtain *k* **where** *a21*: $k \geq 0 \wedge k<((length\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(Suc\ k))) \wedge final\ (l!(Suc\ k))$
by *auto*
then have *a00*:*Suc k<length l* **by** *fastforce*
then obtain *j* **where** *before-k-all-evnt*:$j \leq k \wedge (\Gamma \vdash_c (l!j) \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma \vdash_c (l!k) \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma \vdash_c (cj,sj) \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have $((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a))$
proof −
have *j-length*:$Suc\ j < length\ l$ **using** *a00 before-k-all-evnt* **by** *fastforce*

then have *k-basic*:$cj = (DynCom\ c1) \wedge sj \in Normal\ `\ (p)$
using *a2 pair-j before-k-all-evnt cp env-tran-right assum stability*[*of p R l 0 j j* Γ]
by *force*
then obtain s' **where** *ss*:$sj = Normal\ s' \wedge s' \in (p)$ **by** *auto*
then have *ssj-normal-s*:$ssj = Normal\ s'$
using *before-k-all-evnt k-basic pair-j a0*
by (*metis snd-conv stepc-Normal-elim-cases*(*10*))
have *cj*:$csj=c1\ s'$ **using** *k-basic pair-j ss a0*
by (*metis fst-conv stepc-Normal-elim-cases*(*10*))
moreover have *p1*:$s' \in p$ **using** *ss* **by** *blast*
moreover then have $cpn\ n\ \Gamma\ csj\ ssj \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
using *a0 com-validityn-def cj* **by** *blast*
moreover then have $l!(Suc\ j) = (csj,\ Normal\ s')$
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:$((\Gamma,\ drop\ (Suc\ j)\ l)) \in comm(G,\ (q,a))\ F$
using *j-length a10 a11* Γ*1 ssj-normal-s*
by (*meson contra-subsetD cpn-assum-induct*)
thus *?thesis*
using *j-length l-f drop-comm a10′* Γ*1 cptn-comm-induct*[*of* Γ *l DynCom c1 s - Suc j G q a F Suc j*] *valid*
by *blast*
qed
} **thus** *?thesis* **by** *auto*
qed
note *res* = *conjI* [*OF concl concr*]}
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *force* **qed**
} **thus** *?thesis* **by** *auto* **qed**
} **thus** *?thesis* **by** (*auto simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)

qed

lemma *Guard-sound*:
$\Gamma,\Theta \vdash_{/F}$ *c1 sat* $[p \cap g,\ R,\ G,\ q,a] \Longrightarrow$
$(\forall n.\ \Gamma,\Theta \models n_{/F}$ *c1 sat* $[p \cap g,\ R,\ G,\ q,a]) \Longrightarrow$
Sta $(p \cap g)\ R \Longrightarrow (\forall s.\ (Normal\ s,\ Normal\ s) \in G) \Longrightarrow$
$\Gamma,\Theta \models n_{/F}$ (*Guard f g c1*) *sat* $[p \cap g,\ R,\ G,\ q,a]$
proof −
assume
a0:$\Gamma,\Theta \vdash_{/F}$ *c1 sat* $[(p \cap g)\ ,\ R,\ G,\ q,a]$ **and**
a1:$(\forall n.\ \Gamma,\Theta \models n_{/F}$ *c1 sat* $[p \cap g,\ R,\ G,\ q,a])$ **and**
a2: *Sta* $(p \cap g)\ R$ **and**
a3: $\forall s.\ (Normal\ s,\ Normal\ s) \in G$
{
fix *s*
assume *all-call*:$\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models n_{/F}$ (*Call c*) *sat* $[p,\ R,\ G,\ q,a]$
then have *a1*:$\Gamma \models n_{/F}$ *c1 sat* $[p \cap g,\ R,\ G,\ q,a]$
using *a1 com-cvalidityn-def* **by** *fastforce*
have *cpn n* Γ (*Guard f g c1*) $s \cap assum(p \cap g,\ R) \subseteq comm(G,\ (q,a))\ F$
proof −
{
fix *c*
assume *a10*:$c \in cpn\ n\ \Gamma$ (*Guard f g c1*) *s* **and** *a11*:$c \in assum(p \cap g,\ R)$
then have *a10'*:$c \in cp\ \Gamma$ (*Guard f g c1*) *s*
unfolding *cpn-def cp-def* **using** *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod*
by *fastforce*
obtain $\Gamma 1\ l$ **where** *c-prod*:$c=(\Gamma 1,l)$ **by** *fastforce*
have $c \in comm(G,\ (q,a))\ F$
proof −
{assume *l-f*:*snd* (*last l*) $\notin$ *Fault* ‘ *F*
have *cp*:$l!0=((Guard\ f\ g\ c1),s) \wedge (\Gamma,l) \in cptn \wedge \Gamma=\Gamma 1$ **using** *a10' cp-def*
c-prod **by** *fastforce*
have $\Gamma 1$:$(\Gamma,\ l) = c$ **using** *c-prod cp* **by** *blast*
have *assum*:$snd(l!0) \in Normal$ ‘ $(p \cap g) \wedge (\forall i.\ Suc\ i<length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \rightarrow_e (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have *env-tran*:*env-tran* Γ $(p \cap g)$ *l R* **using** *env-tran-def cp* **by** *blast*
then have *env-tran-right*: *env-tran-right* Γ *l R*
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have *concl*:$(\forall i.\ Suc\ i<length\ l \longrightarrow$
$\Gamma 1\vdash_c(l!i)\ \rightarrow (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof −
{ fix *k ns ns'*
assume *a00*:$Suc\ k<length\ l$ **and**
a21:$\Gamma\vdash_c(l!k)\ \rightarrow (l!(Suc\ k))$
obtain *j* **where** *before-k-all-evnt*:$j\leq k \wedge\ (\Gamma\vdash_c(l!j)\ \rightarrow (l!(Suc\ j))) \wedge (\forall k$

$< j$. $(\Gamma\vdash_c(l!k) \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma\vdash_c(cj,sj) \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have *k-basic*:$cj = (Guard\ f\ g\ c1) \wedge sj \in Normal$ ‘ $(p \cap g)$
using *pair-j before-k-all-evnt cp env-tran-right a2 assum a00 stability*[*of* $p \cap g\ R\ l\ 0\ j\ j\ \Gamma$]
by *force*
then obtain s' **where** *ss*:$sj = Normal\ s' \wedge s' \in (p \cap g)$ **by** *auto*
then have *ssj-normal-s*:$ssj = Normal\ s'$
using *before-k-all-evnt k-basic pair-j a0 stepc-Normal-elim-cases*(*2*)
by (*metis* (*no-types*, *lifting*) *IntD2 prod.inject*)
have $(snd(l!k), snd(l!(Suc\ k))) \in G$
using *ss a2* **unfolding** *Satis-def*
proof (*cases k=j*)
case *True*
have $(Normal\ s', Normal\ s') \in G$ **using** *a3* **by** *auto*
thus $(snd\ (l\ !\ k), snd\ (l\ !\ Suc\ k)) \in G$
using *pair-j k-basic True ss ssj-normal-s* **by** *auto*
next
case *False*
have *j-k*:$j<k$ **using** *before-k-all-evnt False* **by** *fastforce*
thus $(snd\ (l\ !\ k), snd\ (l\ !\ Suc\ k)) \in G$
proof −
have *j-length*:$Suc\ j < length\ l$ **using** *a00 before-k-all-evnt* **by** *fastforce*
have *cj*:$csj=c1$ **using** *k-basic pair-j ss a0*
by (*metis* (*no-types*, *lifting*) *IntD2 fst-conv stepc-Normal-elim-cases*(*2*))

moreover have *p1*:$s' \in (p \cap g)$ **using** *ss* **by** *blast*
moreover then have $cpn\ n\ \Gamma\ csj\ ssj \cap assum(p \cap g, R) \subseteq comm(G, (q,a))\ F$
using *a1 com-validityn-def cj* **by** *blast*
moreover then have $l!(Suc\ j) = (csj, Normal\ s')$
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:$((\Gamma, drop\ (Suc\ j)\ l)) \in comm(G, (q,a))\ F$
using *j-length a10 a11* $\Gamma 1$ *ssj-normal-s*
cpn-assum-induct
by *fastforce*
then show *?thesis*
using *a00 a21 a10'* $\Gamma 1$ *j-k j-length l-f*
cptn-comm-induct[*of* $\Gamma\ l\ (Guard\ f\ g\ c1)\ s$ - $Suc\ j\ G\ q\ a\ F\ k$]
unfolding *Satis-def* **by** *fastforce*
qed
qed
} **thus** *?thesis* **by** (*simp add*: *c-prod cp*) **qed**
have *concr*:(*final* (*last l*) $\longrightarrow$
$((fst\ (last\ l) = Skip\ \wedge$

snd (*last l*) ∈ *Normal* ' *q*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ' (*a*)))
proof−
{
assume *valid*:*final* (*last l*)
have $\exists k.\ k \geq 0 \wedge k < ((\mathit{length}\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(\mathit{Suc}\ k))) \wedge \mathit{final}\ (l!(\mathit{Suc}\ k))$
proof −
have *len-l*:*length l* > *0* **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:*l*=*a1*#*l1* **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:*last l* = *l*!(*length l*−*1*)
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
have *final-0*:¬*final*(*l*!*0*) **using** *cp* **unfolding** *final-def* **by** *auto*
have *0*≤ (*length l*−*1*) **using** *len-l last-l* **by** *auto*
moreover have (*length l*−*1*) < *length l* **using** *len-l* **by** *auto*
moreover have *final* (*l*!(*length l*−*1*)) **using** *valid last-l* **by** *auto*
moreover have *fst* (*l*!*0*) = (*Guard f g c1*) **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-final env-tran-right final-0*
by *blast*
qed
then obtain *k* **where** *a21*: $k \geq 0 \wedge k < ((\mathit{length}\ l) - 1) \wedge (\Gamma \vdash_c (l!k) \rightarrow (l!(\mathit{Suc}\ k))) \wedge \mathit{final}\ (l!(\mathit{Suc}\ k))$
by *auto*
then have *a00*:*Suc k*<*length l* **by** *fastforce*
then obtain *j* **where** *before-k-all-evnt*:$j \leq k \wedge (\Gamma \vdash_c (l!j) \rightarrow (l!(\mathit{Suc}\ j))) \wedge (\forall k < j.\ (\Gamma \vdash_c (l!k) \rightarrow_e (l!(\mathit{Suc}\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma \vdash_c (cj,sj) \rightarrow (csj,ssj)) \wedge cj = \mathit{fst}\ (l!j) \wedge sj = \mathit{snd}\ (l!j) \wedge csj = \mathit{fst}\ (l!(\mathit{Suc}\ j)) \wedge ssj = \mathit{snd}(l!(\mathit{Suc}\ j))$
by *fastforce*
have ((*fst* (*last l*) = *Skip* ∧
snd (*last l*) ∈ *Normal* ' *q*)) ∨
(*fst* (*last l*) = *Throw* ∧
snd (*last l*) ∈ *Normal* ' (*a*))
proof −
have *j-length*:*Suc j* < *length l* **using** *a00 before-k-all-evnt* **by** *fastforce*

then have *k-basic*:*cj* = (*Guard f g c1*) ∧ *sj* ∈ *Normal* ' (*p* ∩ *g*)
using *pair-j before-k-all-evnt cp env-tran-right a2 assum a00 stability*[*of p* ∩ *g R l 0 j j* Γ]
by *force*
then obtain *s'* **where** *ss*:*sj* = *Normal s'* ∧ *s'*∈ (*p* ∩ *g*) **by** *auto*
then have *ssj-normal-s*:*ssj* = *Normal s'*
using *before-k-all-evnt k-basic pair-j a1*
by (*metis* (*no-types*, *lifting*) *IntD2 Pair-inject stepc-Normal-elim-cases*(*2*))

have *cj*:*csj*=*c1* **using** *k-basic pair-j ss a0*
by (*metis* (*no-types*, *lifting*) *fst-conv IntD2 stepc-Normal-elim-cases*(*2*))

moreover have *p1*:$s' \in (p \cap g)$ **using** *ss* **by** *blast*
moreover then have *cpn n* Γ *csj ssj* ∩ *assum*((*p* ∩ *g*), *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*
using *a1 com-validityn-def cj* **by** *blast*
moreover then have *l*!(*Suc j*) = (*csj*, *Normal s'*)
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:((Γ, *drop* (*Suc j*) *l*))∈ *comm*(*G*, (*q*,*a*)) *F*
using *j-length a10 a11* Γ*1 ssj-normal-s cpn-assum-induct*
by *fastforce*
thus *?thesis*
using *j-length l-f drop-comm a10'* Γ*1 cptn-comm-induct*[*of* Γ *l* (*Guard f g c1*) *s* - *Suc j G q a F Suc j*] *valid*
by *blast*
qed
} **thus** *?thesis* **by** *auto*
qed
note *res* = *conjI* [*OF concl concr*]**}**
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *force* **qed**
} **thus** *?thesis* **by** *auto* **qed**
} **thus** *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Guarantee-sound*:
$\Gamma,\Theta \vdash_{/F}$ *c1 sat* [(*p* ∩ *g*), *R*, *G*, *q*,*a*] ⟹
$\forall n.\ \Gamma,\Theta \models n_{/F}$ *c1 sat* [*p* ∩ *g*, *R*, *G*, *q*,*a*] ⟹
Sta p R ⟹
f∈*F* ⟹
(∀ *s*. (*Normal s*, *Normal s*) ∈ *G*) ⟹
$\Gamma,\Theta \models n_{/F}$ (*Guard f g c1*) *sat* [*p*, *R*, *G*, *q*,*a*]
proof −
assume
a0:$\Gamma,\Theta \vdash_{/F}$ *c1 sat* [*p* ∩ *g*, *R*, *G*, *q*,*a*] **and**
a1:$\forall n.\ \Gamma,\Theta \models n_{/F}$ *c1 sat* [*p* ∩ *g*, *R*, *G*, *q*,*a*] **and**
a2: *Sta p R* **and**
a3: (∀ *s*. (*Normal s*, *Normal s*) ∈ *G*) **and**
a4: *f*∈*F*
{
fix *s*
assume *all-call*:∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. $\Gamma \models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]
then have *a1*:$\Gamma \models n_{/F}$ *c1 sat* [*p* ∩ *g*, *R*, *G*, *q*,*a*]
using *a1 com-cvalidityn-def* **by** *fastforce*
have *cpn n* Γ (*Guard f g c1*) *s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*
proof −

{
fix *c*
assume $a10{:}c \in cpn\ n\ \Gamma\ (Guard\ f\ g\ c1)\ s$ **and** $a11{:}c \in assum(p,\ R)$
then have $a10'{:}c \in cp\ \Gamma\ (Guard\ f\ g\ c1)\ s$
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fast*
obtain $\Gamma 1\ l$ **where** $c\text{-}prod{:}c{=}(\Gamma 1,l)$ **by** *fastforce*
have $c \in comm(G,\ (q,a))\ F$
proof −
{**assume** $l\text{-}f{:}snd\ (last\ l) \notin Fault\ `\ F$
have $cp{:}l!0{=}((Guard\ f\ g\ c1),s) \wedge (\Gamma,l) \in cptn \wedge \Gamma{=}\Gamma 1$ **using** $a10'$ *cp-def c-prod* **by** *fastforce*
have $\Gamma 1{:}(\Gamma,\ l) = c$ **using** *c-prod cp* **by** *blast*
have $assum{:}snd(l!0) \in Normal\ `\ (p) \wedge (\forall i.\ Suc\ i{<}length\ l \longrightarrow$
$(\Gamma 1)\vdash_c(l!i)\ \rightarrow_e\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in R)$
using *a11 c-prod* **unfolding** *assum-def* **by** *simp*
then have $env\text{-}tran{:}env\text{-}tran\ \Gamma\ p\ l\ R$ **using** *env-tran-def cp* **by** *blast*
then have $env\text{-}tran\text{-}right{:}\ env\text{-}tran\text{-}right\ \Gamma\ l\ R$
using *env-tran env-tran-right-def* **unfolding** *env-tran-def* **by** *auto*
have $concl{:}(\forall i\ ns\ ns'.\ Suc\ i{<}length\ l \longrightarrow$
$\Gamma 1\vdash_c(l!i)\ \rightarrow\ (l!(Suc\ i)) \longrightarrow$
$(snd(l!i),\ snd(l!(Suc\ i))) \in G)$
proof −
{ **fix** $k\ ns\ ns'$
assume $a00{:}Suc\ k{<}length\ l$ **and**
$a21{:}\Gamma\vdash_c(l!k)\ \rightarrow\ (l!(Suc\ k))$
obtain *j* **where** $before\text{-}k\text{-}all\text{-}evnt{:}j{\le}k \wedge\ (\Gamma\vdash_c(l!j)\ \rightarrow\ (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma\vdash_c(l!k)\ \rightarrow_e\ (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain $cj\ sj\ csj\ ssj$ **where** $pair\text{-}j{:}(\Gamma\vdash_c(cj,sj)\ \rightarrow\ (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have $k\text{-}basic{:}cj\ {=}(Guard\ f\ g\ c1) \wedge sj \in Normal\ `\ (p)$
using *pair-j before-k-all-evnt cp env-tran-right a2 assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain s' **where** $ss{:}sj = Normal\ s' \wedge s'{\in}\ (p)$ **by** *auto*
have $or{:}s'{\in}\ (g \cup (-g))$ **by** *fastforce*
{**assume** $s' \in g$
then have $k\text{-}basic{:}cj\ {=}(Guard\ f\ g\ c1) \wedge sj \in Normal\ `\ (p \cap g)$
using *ss k-basic* **by** *fastforce*
then have $ss{:}\ sj = Normal\ s' \wedge s'{\in}\ (p \cap g)$
using *ss* **by** *fastforce*
have $ssj\text{-}normal\text{-}s{:}ssj = Normal\ s'$
using *ss before-k-all-evnt k-basic pair-j a0 stepc-Normal-elim-cases*(*2*)
by (*metis* (*no-types*, *lifting*) *IntD2 prod.inject*)
have $(snd(l!k),\ snd(l!(Suc\ k))) \in G$
using *ss a2* **unfolding** *Satis-def*

```
proof (cases k=j)
  case True
  have (Normal s', Normal s') ∈ G using a3 by auto
  thus (snd (l ! k), snd (l ! Suc k)) ∈  G
    using pair-j k-basic True ss ssj-normal-s by auto
next
  case False
 have j-k:j<k using  before-k-all-evnt False by fastforce
  thus (snd (l ! k), snd (l ! Suc k)) ∈  G
  proof −
    have j-length:Suc j < length l using a00 before-k-all-evnt by fastforce
    have cj:csj=c1 using k-basic pair-j ss a0
   by (metis (no-types, lifting) fst-conv IntD2 stepc-Normal-elim-cases(2))

    moreover have p1:s' ∈ (p ∩ g) using ss by blast
   moreover then have cpn n Γ csj ssj ∩ assum((p ∩ g), R) ⊆ comm(G,
(q,a)) F
      using a1 com-validityn-def cj by blast
    moreover then have l!(Suc j) = (csj, Normal s')
      using before-k-all-evnt pair-j cj ssj-normal-s
      by fastforce
   ultimately have drop-comm:((Γ, drop (Suc j) l))∈ comm(G, (q,a)) F
      using  j-length a10 a11 Γ1  ssj-normal-s cpn-assum-induct
      by fastforce
     then show ?thesis
     using a3 a00 a21  a10' Γ1  j-k j-length l-f
      cptn-comm-induct[of Γ l (Guard f g c1) s - Suc j G q a F k]
     unfolding Satis-def by fastforce
   qed
 qed
} note p1=this
{
 assume s'∉g
 then have csj-skip:csj= Skip ∧ ssj=Fault f using k-basic ss pair-j
   by (meson Pair-inject stepc-Normal-elim-cases(2))
 then have snd (last l) = Fault f using pair-j
 proof −
   have j = k
   proof −
     have f1: k < length l
       using a00 by linarith
     have ¬ SmallStepCon.final (l ! k)
       by (metis SmallStepCon.no-step-final' a21)
     then have ¬ Suc j ≤ k
         using f1 SmallStepCon.final-def cp csj-skip i-skip-all-skip pair-j by
blast
     then show ?thesis
       by (metis Suc-leI before-k-all-evnt le-eq-less-or-eq)
   qed
```

then have *False*
using *pair-j csj-skip* **by** (*metis a00 a4 cp image-eqI l-f last-not-F*)
then show *?thesis*
by *metis*
qed
then have *False* **using** *a4 l-f* **by** *auto*
}
then have $(snd(l!k),\ snd(l!(Suc\ k))) \in G$
using *p1 or* **by** *fastforce*
} thus *?thesis* **by** (*simp add*: *c-prod cp*) **qed**
have *concr*:$(final\ (last\ l)\ \longrightarrow$
$((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)))$
proof−
{
assume *valid*:*final* (*last l*)
have $\exists k.\ k \geq 0 \wedge k<((length\ l) - 1) \wedge (\Gamma\vdash_c(l!k)\ \rightarrow (l!(Suc\ k))) \wedge final\ (l!(Suc\ k))$
proof −
have *len-l*:$length\ l > 0$ **using** *cp* **using** *cptn.simps* **by** *blast*
then obtain *a1 l1* **where** *l*:$l=a1\#l1$ **by** (*metis SmallStepCon.nth-tl length-greater-0-conv*)
have *last-l*:$last\ l = l!(length\ l-1)$
using *last-length* [*of a1 l1*] *l* **by** *fastforce*
have *final-0*:$\neg final(l!0)$ **using** *cp* **unfolding** *final-def* **by** *auto*
have $0\leq (length\ l-1)$ **using** *len-l last-l* **by** *auto*
moreover have $(length\ l-1) < length\ l$ **using** *len-l* **by** *auto*
moreover have $final\ (l!(length\ l-1))$ **using** *valid last-l* **by** *auto*
moreover have $fst\ (l!0) = (Guard\ f\ g\ c1)$ **using** *cp* **by** *auto*
ultimately show *?thesis*
using *cp final-exist-component-tran-final env-tran-right final-0*
by *blast*
qed
then obtain *k* **where** *a21*: $k\geq 0 \wedge k<((length\ l) - 1) \wedge (\Gamma\vdash_c(l!k)\ \rightarrow (l!(Suc\ k))) \wedge final\ (l!(Suc\ k))$
by *auto*
then have *a00*:$Suc\ k<length\ l$ **by** *fastforce*
then obtain *j* **where** *before-k-all-evnt*:$j\leq k \wedge\ (\Gamma\vdash_c(l!j)\ \rightarrow (l!(Suc\ j))) \wedge (\forall k < j.\ (\Gamma\vdash_c(l!k)\ \rightarrow_e (l!(Suc\ k))))$
using *a00 a21 exist-first-comp-tran cp* **by** *blast*
then obtain *cj sj csj ssj* **where** *pair-j*:$(\Gamma\vdash_c(cj,sj)\ \rightarrow (csj,ssj)) \wedge cj = fst\ (l!j) \wedge sj = snd\ (l!j) \wedge csj = fst\ (l!(Suc\ j)) \wedge ssj = snd(l!(Suc\ j))$
by *fastforce*
have $((fst\ (last\ l) = Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)) \vee$
$(fst\ (last\ l) = Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a))$

proof −
have *j-length*:*Suc j* < *length l* **using** *a00 before-k-all-evnt* **by** *fastforce*

have *k-basic*:*cj* =(*Guard f g c1*) ∧ *sj* ∈ *Normal* ‘ (*p*)
using *pair-j before-k-all-evnt cp env-tran-right a2 assum a00 stability*[*of p R l 0 j j* Γ]
by *force*
then obtain *s′* **where** *ss*:*sj* = *Normal s′* ∧ *s′*∈ (*p*) **by** *auto*
have *or*:*s′*∈ (*g* ∪ (−*g*)) **by** *fastforce*
{**assume** *s′* ∈ *g*
then have *k-basic*:*cj* =(*Guard f g c1*) ∧ *sj* ∈ *Normal* ‘ (*p* ∩ *g*)
using *ss k-basic* **by** *fastforce*
then have *ss*: *sj* = *Normal s′* ∧ *s′*∈ (*p* ∩ *g*)
using *ss* **by** *fastforce*
then have *ssj-normal-s*:*ssj* = *Normal s′*
using *before-k-all-evnt k-basic pair-j a1*
by (*metis* (*no-types*, *lifting*) *Pair-inject IntD2 stepc-Normal-elim-cases*(*2*))

have *cj*:*csj*=*c1* **using** *k-basic pair-j ss a0*
by (*metis* (*no-types*, *lifting*) *fst-conv IntD2 stepc-Normal-elim-cases*(*2*))

moreover have *p1*:*s′*∈(*p* ∩ *g*) **using** *ss* **by** *blast*
moreover then have *cpn n* Γ *csj ssj* ∩ *assum*((*p* ∩ *g*), *R*) ⊆ *comm*(*G*, (*q*,*a*)) *F*
using *a1 com-validityn-def cj* **by** *blast*
moreover then have *l*!(*Suc j*) = (*csj*, *Normal s′*)
using *before-k-all-evnt pair-j cj ssj-normal-s*
by *fastforce*
ultimately have *drop-comm*:((Γ, *drop* (*Suc j*) *l*))∈ *comm*(*G*, (*q*,*a*)) *F*
using *j-length a10 a11* Γ*1* *ssj-normal-s cpn-assum-induct*
by *fastforce*
then have *?thesis*
using *j-length l-f drop-comm a10′* Γ*1*
cptn-comm-induct[*of* Γ *l* (*Guard f g c1*) *s - Suc j G q a F Suc j*] *valid*
by *blast*
}**note** *left*=*this*
{

assume *s′*∉*g*
then have *csj*= *Skip* ∧ *ssj*=*Fault f* **using** *k-basic ss pair-j*
by (*meson Pair-inject stepc-Normal-elim-cases*(*2*))
then have *snd* (*last l*) = *Fault f* **using** *pair-j*
by (*metis a4 cp imageI j-length l-f last-not-F*)
then have *False* **using** *a4 l-f* **by** *auto*
}
thus *?thesis* **using** *or left* **by** *auto* **qed**
} **thus** *?thesis* **by** *auto*
qed

note *res* = *conjI* [*OF concl concr*]}
thus *?thesis* **using** *c-prod* **unfolding** *comm-def* **by** *force* **qed**
} **thus** *?thesis* **by** *auto* **qed**
} **thus** *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *WhileNone*:
$\Gamma\vdash_c$ (*While b c1*, *s1*) → (*LanguageCon.com.Skip*, *t1*) ⟹
(*n*,Γ, (*Skip*, *t1*) # *xsa*) ∈ *cptn-mod-nest-call* ⟹
Γ $\models n_{/F}$ *c1 sat* [*p* ∩ *b*,*R*, *G*, *p*,*a*] ⟹
Sta p R ⟹
Sta (*p* ∩ (−*b*)) *R* ⟹
Sta a R ⟹
(∀ *s*. (*Normal s*, *Normal s*) ∈ *G*) ⟹
(Γ, (*While b c1*, *s1*) # (*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈ *assum* (*p*, *R*) ⟹
(Γ, (*While b c1*, *s1*) # (*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈ *comm* (*G*,(*p* ∩ (−*b*)),*a*) *F*
proof −
assume *a0*:$\Gamma\vdash_c$ (*While b c1*, *s1*) → (*LanguageCon.com.Skip*, *t1*) **and**
a1:(*n*,Γ, (*Skip*, *t1*) # *xsa*) ∈ *cptn-mod-nest-call* **and**
a2: Γ $\models n_{/F}$ *c1 sat* [*p* ∩ *b*,*R*, *G*, *p*,*a*] **and**
a3:*Sta p R* **and**
a4:*Sta* (*p* ∩ (−*b*)) *R* **and**
a5:*Sta a R* **and**
a6:∀ *s*. (*Normal s*, *Normal s*) ∈ *G* **and**
a7:(Γ, (*While b c1*, *s1*) # (*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈ *assum* (*p*, *R*)
obtain *s1'* **where** *s1N*:*s1*=*Normal s1'* ∧ *s1'*∈*p* **using** *a7* **unfolding** *assum-def* **by** *fastforce*
then have *s1-t1*:*s1'*∉ *b* ∧ *t1*=*s1* **using** *a0*
using *LanguageCon.com.distinct*(*5*) *prod.inject*
by (*fastforce elim*:*stepc-Normal-elim-cases*(*7*))
then have *t1-Normal-post*:*t1*∈ *Normal* ‘ (*p* ∩ (−*b*))
using *s1N* **by** *fastforce*
also have (Γ, (*While b c1*, *s1*) # (*LanguageCon.com.Skip*, *t1*) # *xsa*)∈*cptn*
using *a1 a0 cptn.simps*
using *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
ultimately have *assum-skip*:
(Γ,(*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈ *assum* ((*p* ∩ (−*b*)), *R*)
using *a1 a7 tl-of-assum-in-assum1 t1-Normal-post* **by** *fastforce*
have *skip-comm*:(Γ,(*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈
comm (*G*,((*p* ∩ (−*b*)),*a*)) *F*
proof−
obtain Θ **where** (∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. Γ $\models n_{/F}$ (*Call c*) *sat* [*p* , *R*, *G*, *q*,*a*]) **by** *auto*
moreover have Γ,Θ $\models n_{/F}$ *Skip sat* [(*p* ∩ (−*b*)), *R*, *G*, (*p* ∩ (−*b*)),*a*]
using *Skip-sound*[*of* (*p* ∩ − *b*)] *a4 a6* **by** *blast*
ultimately show *?thesis*

using *assum-skip a1* **unfolding** *com-cvalidityn-def com-validityn-def cpn-def*
by *fastforce*
qed
have *G-ref*:(*Normal s1'*, *Normal s1'*)∈*G* **using** *a6* **by** *fastforce*
thus *?thesis* **using** *skip-comm ctran-in-comm*[*of s1'*] *s1N s1-t1* **by** *blast*
qed

lemma *while1*:
(n,Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *cptn-mod-nest-call* ⟹
s1 ∈ *b* ⟹
xsa = *map* (*lift* (*While b c*)) *xs1* ⟹
Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*,*R*, *G*, *p*,*a*] ⟹
(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*LanguageCon.com.While b c*), *Normal s1*) # *xsa*)
∈ *assum* (*p*, *R*) ⟹
∀ *s*. (*Normal s*, *Normal s*) ∈ *G* ⟹
(Γ, (*LanguageCon.com.While b c*, *Normal s1*) #
(*LanguageCon.com.Seq c* (*LanguageCon.com.While b c*), *Normal s1*) #
xsa)
∈ *comm* (*G*, *p*∩(−*b*), *a*) *F*
proof −
assume
a0:(n,Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *cptn-mod-nest-call* **and**
a1:*s1* ∈ *b* **and**
a2:*xsa* = *map* (*lift* (*While b c*)) *xs1* **and**
a3:Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*,*R*, *G*, *p*,*a*] **and**
a4:(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*)
∈ *assum* (*p*, *R*) **and**
a5:∀ *s*. (*Normal s*,*Normal s*) ∈ *G*
have *seq-map*:(*Seq c* (*While b c*), *Normal s1*) # *xsa*=
map (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*)
using *a2* **unfolding** *lift-def* **by** *fastforce*
have *step*:$\Gamma\vdash_c$(*While b c*,*Normal s1*) → (*Seq c* (*While b c*),*Normal s1*) **using** *a1*
WhileTruec **by** *fastforce*
have *s1-normal*:*s1* ∈ *p* ∧ *s1* ∈ *b* **using** *a4 a1* **unfolding** *assum-def* **by** *fastforce*
then have *G-ref*:(*Normal s1*, *Normal s1*) ∈ *G* **using** *a5* **by** *fastforce*
have *s1-collect-p*: *Normal s1*∈ *Normal* ‘ (*p* ∩ *b*) **using** *s1-normal* **by** *fastforce*
have (Γ, *map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))∈*cptn*
using *a2 cptn-eq-cptn-mod-nest lift-is-cptn a0* **by** *blast*
then have *cptn-seq*:(Γ,(*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈*cptn*
using *seq-map* **by** *auto*
then have (Γ, (*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *xsa*)
∈ *cptn*
using *step* **by** (*simp add*: *cptn.CptnComp*)
then have *assum-seq*:(Γ,(*Seq c* (*While b c*), *Normal s1*) # *xsa*)∈*assum* (*p*, *R*)
using *a4 tl-of-assum-in-assum1 s1-collect-p* **by** *fastforce*
have *cp-c*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cpn n* Γ *c* (*Normal s1*))
using *a0* **unfolding** *cpn-def* **by** *fastforce*

then have *cp-c'*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cp* Γ *c* (*Normal s1*))
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-nest* **by** *fastforce*
also have *cp-seq*:(Γ, (*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈ (*cp* Γ (*Seq c* (*While b c*)) (*Normal s1*))
using *cptn-seq* **unfolding** *cp-def* **by** *fastforce*
ultimately have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*(*p*,*R*)
using *assum-map assum-seq seq-map* **by** *fastforce*
then have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*((*p* ∩ *b*),*R*)
unfolding *assum-def* **using** *s1-collect-p* **by** *fastforce*
then have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *a3 cp-c* **unfolding** *com-validityn-def* **by** *fastforce*
then have (Γ, (*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *cp-seq cp-c' comm-map seq-map* **by** *fastforce*
then have (Γ, (*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *G-ref ctran-in-comm* **by** *fastforce*
also have ¬ *final* (*last* ((*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *xsa*))
using *seq-map* **unfolding** *final-def lift-def* **by** (*simp add*: *case-prod-beta' last-map*)
ultimately show *?thesis* **using** *not-final-in-comm*[*of* Γ] **by** *blast*
qed

lemma *while2*:
(*n*,Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈*cptn-mod-nest-call* ⟹
(*n*, Γ, (*c*, *Normal s1*) # *xs1*) ∈ *cptn-mod-nest-call* ⟹
fst (*last* ((*c*, *Normal s1*) # *xs1*)) = *LanguageCon.com.Skip* ⟹
s1 ∈ *b* ⟹
xsa = *map* (*lift* (*While b c*)) *xs1* @
(*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys* ⟹
(*n*, Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *cptn-mod-nest-call* ⟹
(Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] ⟹
(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *assum* (*p*, *R*) ⟹
(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *comm* (*G*, *p* ∩ (−*b*), *a*) *F*) ⟹
Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] ⟹
(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*)
∈ *assum* (*p*, *R*) ⟹
∀ *s*. (*Normal s*,*Normal s*) ∈ *G* ⟹
(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*)
∈ *comm* (*G*,(*p* ∩ (−*b*), *a*)) *F*
proof −
assume *a00*:(*n*, Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*) ∈*cptn-mod-nest-call* **and**

a0:$(n,\Gamma$, (*c*, *Normal s1*) # *xs1*) ∈ *cptn-mod-nest-call* **and**
a1: *fst* (*last* ((*c*, *Normal s1*) # *xs1*)) = *LanguageCon.com.Skip* **and**
a2:*s1* ∈ *b* **and**
a3:*xsa* = *map* (*lift* (*While b c*)) *xs1* @
(*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys* **and**
a4:$(n,\Gamma$, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *cptn-mod-nest-call* **and**
a5:$\Gamma \models n_{/F}$ *c sat* [$p \cap b$, *R*, *G*, *p*,*a*] **and**
a6:(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*)
∈ *assum* (*p*, *R*) **and**
a7:($\Gamma \models n_{/F}$ *c sat* [$p \cap b$, *R*, *G*, *p*,*a*] ⟹
(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *assum* (*p*, *R*) ⟹
(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *comm* (*G*,$p \cap (-b)$, *a*) *F*) **and**
a8:∀ *s*. (*Normal s*, *Normal s*) ∈ *G*
let *?l*= (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) # *xsa*
let *?sub-l*=((*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*)
{
assume *final-not-fault*:*snd* (*last ?l*) ∉ *Fault* ‘ *F*
have *a0'*:(Γ, (*c*, *Normal s1*) # *xs1*) ∈ *cptn*
using *cptn-eq-cptn-mod-nest* **using** *a0* **by** *auto*
have *a4*:(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*) ∈ *cptn*
using *cptn-eq-cptn-mod-nest* **using** *a4* **by** *blast*
have *seq-map*:(*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*=
map (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*)
using *a2* **unfolding** *lift-def* **by** *fastforce*
have *step*:$\Gamma\vdash_c$(*While b c*,*Normal s1*) → (*Seq c* (*While b c*),*Normal s1*) **using** *a2*
WhileTruec **by** *fastforce*
have *s1-normal*:*s1*∈*p* ∧ *s1* ∈ *b* **using** *a6 a2* **unfolding** *assum-def* **by** *fastforce*
have *G-ref*:(*Normal s1*, *Normal s1*)∈*G*
using *a8* **by** *blast*
have *s1-collect-p*: *Normal s1*∈ *Normal* ‘ ($p \cap b$) **using** *s1-normal* **by** *fastforce*
have (Γ, *map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))∈*cptn*
using *a2 cptn-eq-cptn-mod lift-is-cptn a0'* **by** *fastforce*
then have *cptn-seq*:(Γ,(*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*))
xs1) ∈*cptn*
using *seq-map* **by** *auto*
then have (Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*) ∈ *cptn*
using *step* **by** (*simp add*: *cptn.CptnComp*)
also have (Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*)

$\in$ *assum* (*p*, *R*)
using *a6 a3 sub-assum* **by** *force*
ultimately have *assum-seq*:(Γ,(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*) ∈ *assum* (*p*, *R*)
using *a6 tl-of-assum-in-assum1 s1-collect-p*
tl-of-assum-in-assum **by** *fastforce*
have *cpn-c*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cpn n* Γ *c* (*Normal s1*))
using *a0* **unfolding** *cpn-def* **by** *fastforce*
have *cp-c*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cp* Γ *c* (*Normal s1*))
using *a0′* **unfolding** *cp-def* **by** *fastforce*
also have *cp-seq*:(Γ, (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ (*cp* Γ (*Seq c* (*While b c*)) (*Normal s1*))
using *cptn-seq* **unfolding** *cp-def* **by** *fastforce*
ultimately have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*(*p*,*R*)
using *assum-map assum-seq seq-map* **by** *fastforce*
then have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*((*p* ∩ *b*),*R*)
unfolding *assum-def* **using** *s1-collect-p* **by** *fastforce*
then have *c-comm*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *a5 cpn-c* **unfolding** *com-validityn-def* **by** *fastforce*
then have (Γ, (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *cp-seq cp-c comm-map seq-map* **by** *fastforce*
then have *comm-while*:(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *G-ref ctran-in-comm* **by** *fastforce*
have *final-last-c*:*final* (*last* ((*c*,*Normal s1*)#*xs1*))
using *a1 a3* **unfolding** *final-def* **by** *fastforce*
have *last-while1*:*snd* (*last* (*map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))) = *snd* (*last* ((*c*, *Normal s1*) # *xs1*))
unfolding *lift-def* **by** (*simp add*: *case-prod-beta′ last-map*)
have *last-while2*:(*last* (*map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))) =
last ((*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*)
using *seq-map* **by** *fastforce*
have *not-fault-final-last-c*:
snd (*last* ((*c*,*Normal s1*)#*xs1*)) ∉ *Fault* ‘ *F*
proof −
have (*length ?sub-l*) − *1* < *length ?l*
using *a3* **by** *fastforce*
then have *snd* (*?l*!((*length ?sub-l*) − *1*))∉ *Fault* ‘ *F*
using *final-not-fault a3 a00 last-not-F*[*of* Γ *?l F*]
cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod **by** *blast*
thus *?thesis* **using** *last-while2 last-while1 seq-map*
by (*metis* (*no-types*) *Cons-lift-append a3 diff-Suc-1 last-length length-Cons lessI nth-Cons-Suc nth-append*)
qed
then have *last-c-normal*:*snd* (*last* ((*c*,*Normal s1*)#*xs1*)) ∈ *Normal* ‘ (*p*)
using *c-comm a1* **unfolding** *comm-def final-def* **by** *fastforce*

then obtain *sl* **where** *sl*:*snd* (*last* ((*c*,*Normal s1*)#*xs1*)) = *Normal sl* **by** *fastforce*
have *while-comm*:(Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*) ∈ *comm*(*G*,(*p*∩(−*b*),*a*)) *F*
proof −
have *assum-while*: (Γ, (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) # *ys*)
∈ *assum* (*p*, *R*)
using *last-c-normal a3 a6 sub-assum-r*[*of* Γ *?sub-l* (*While b c*, *snd* (*last* ((*c*, *Normal s1*) # *xs1*))) *ys p R p*]
by *fastforce*
thus *?thesis* **using** *a5 a7* **by** *fastforce*
qed
have *sl*∈*p* **using** *last-c-normal sl* **by** *fastforce*
then have *G1-ref*:(*Normal sl*, *Normal sl*)∈*G* **using** *a8* **by** *auto*
also have *snd* (*last ?sub-l*) = *Normal sl*
using *last-while1 last-while2 sl* **by** *fastforce*
ultimately have *?thesis*
using *cptn-eq-cptn-mod-nest a00 a3 sl while-comm comm-union*[*OF comm-while*]

by *fastforce*
} **note** *p1* =*this*
{
assume *final-not-fault*:¬ (*snd* (*last ?l*) ∉ *Fault* ' *F*)
then have *?thesis* **unfolding** *comm-def* **by** *fastforce*
} **thus** *?thesis* **using** *p1* **by** *fastforce*
qed

lemma *while3*:
(*n*, Γ, (*c*, *Normal s1*) # *xs1*) ∈ *cptn-mod-nest-call* ⟹
fst (*last* ((*c*, *Normal s1*) # *xs1*)) = *Throw* ⟹
s1 ∈ *b* ⟹
snd (*last* ((*c*, *Normal s1*) # *xs1*)) = *Normal sl* ⟹
(*n*,Γ, (*Throw*, *Normal sl*) # *ys*) ∈ *cptn-mod-nest-call* ⟹
Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*,*R*, *G*, *p*,*a*] ⟹
(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
(*map* (*lift* (*While b c*)) *xs1* @
(*Throw*, *Normal sl*) # *ys*))
∈ *assum* (*p*, *R*) ⟹
Sta p R ⟹
Sta a R ⟹ ∀ *s*. (*Normal s*, *Normal s*) ∈ *G* ⟹
(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
((*map* (*lift* (*While b c*)) *xs1* @
(*Throw*, *Normal sl*) # *ys*))) ∈ *comm* (*G*, *p*∩ (−*b*), *a*) *F*

proof −
assume *a0*:(*n*,Γ, (*c*, *Normal s1*) # *xs1*) ∈ *cptn-mod-nest-call* **and**
a1:*fst* (*last* ((*c*, *Normal s1*) # *xs1*)) = *Throw* **and**

a2:*s1* ∈ *b* **and**
a3:*snd* (*last* ((*c*, *Normal s1*) # *xs1*)) = *Normal sl* **and**
a4:(*n*,Γ, (*Throw*, *Normal sl*) # *ys*) ∈ *cptn-mod-nest-call* **and**
a5:Γ $\models n_{/F}$ *c sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] **and**
a6:(Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
(*map* (*lift* (*While b c*)) *xs1* @
(*Throw*, *Normal sl*) # *ys*))
∈ *assum* (*p*, *R*) **and**
a7: *Sta p R* **and**
a8: *Sta a R* **and**
a10:∀ *s*. (*Normal s*,*Normal s*) ∈ *G*
have *seq-map*:(*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*=
map (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*)
using *a2* **unfolding** *lift-def* **by** *fastforce*
have *step*:Γ$\vdash_c$(*While b c*,*Normal s1*) → (*Seq c* (*While b c*),*Normal s1*) **using** *a2* *WhileTruec* **by** *fastforce*
have *s1-normal*:*s1*∈*p* ∧ *s1* ∈ *b* **using** *a6 a2* **unfolding** *assum-def* **by** *fastforce*
then have *G-ref*:(*Normal s1*, *Normal s1*)∈*G* **using** *a10* **by** *auto*
have *s1-collect-p*: *Normal s1*∈ *Normal* ' (*p* ∩ *b*) **using** *s1-normal* **by** *fastforce*
have (*n*, Γ, *map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))∈*cptn-mod-nest-call*
using *a2 lift-is-cptn a0*
by (*metis cptn-mod-nest-call*.*CptnModNestSeq1 seq-map*)
then have *cptn-seq*:(*n*,Γ,(*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈*cptn-mod-nest-call*
using *seq-map* **by** *auto*
then have *cptn*:(*n*,Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*) ∈ *cptn-mod-nest-call*
by (*meson a0 a2 cptn-mod-nest-call*.*CptnModNestWhile1*)
also have (Γ, (*LanguageCon*.*com*.*While b c*, *Normal s1*) #
(*LanguageCon*.*com*.*Seq c* (*LanguageCon*.*com*.*While b c*), *Normal s1*) #
map (*lift* (*LanguageCon*.*com*.*While b c*)) *xs1*)
∈ *assum* (*p*, *R*)
using *a6 sub-assum* **by** *force*
ultimately have *assum-seq*:(Γ,(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*) ∈ *assum* (*p*, *R*)
using *a6 tl-of-assum-in-assum1 s1-collect-p*
tl-of-assum-in-assum cptn-eq-cptn-mod-nest **by** *fast*
have *cpn-c*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cpn n* Γ *c* (*Normal s1*))
using *a0* **unfolding** *cpn-def* **by** *fastforce*
then have *cp-c*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ (*cp* Γ *c* (*Normal s1*))
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-nest* **by** *auto*
moreover have *cp-seq*:(Γ, (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ (*cpn n* Γ (*Seq c* (*While b c*)) (*Normal s1*))
using *cptn-seq* **unfolding** *cpn-def* **by** *fastforce*
then have *cp-seq'*:(Γ, (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ (*cp* Γ (*Seq c* (*While b c*)) (*Normal s1*))
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-nest* **by** *auto*

ultimately have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*(*p*,*R*)
using *assum-map assum-seq seq-map* **by** *fastforce*
then have (Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *assum*((*p* ∩ *b*),*R*)
unfolding *assum-def* **using** *s1-collect-p* **by** *fastforce*
then have *c-comm*:(Γ, ((*c*, *Normal s1*) # *xs1*)) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *a5 cpn-c* **unfolding** *com-validityn-def* **by** *fastforce*
then have (Γ, (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *cp-seq′ cp-c comm-map seq-map* **by** *fastforce*
then have *comm-while*:(Γ, (*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*) ∈ *comm*(*G*,(*p*,*a*)) *F*
using *G-ref ctran-in-comm* **by** *fastforce*
have *final-last-c*:*final* (*last* ((*c*,*Normal s1*)#*xs1*))
using *a1 a3* **unfolding** *final-def* **by** *fastforce*
have *not-fault-final-last-c*:
snd (*last* ((*c*,*Normal s1*)#*xs1*)) ∉ *Fault* ‘ *F*
using *a3* **by** *fastforce*
then have *sl-a*:*Normal sl* ∈ *Normal* ‘ (*a*)
using *final-last-c a1 c-comm* **unfolding** *comm-def*
using *a3 comm-dest2*
by *auto*
have *last-while1*:*snd* (*last* (*map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))) = *snd* (*last* ((*c*, *Normal s1*) # *xs1*))
unfolding *lift-def* **by** (*simp add*: *case-prod-beta′ last-map*)
have *last-while2*:(*last* (*map* (*lift* (*While b c*)) ((*c*,*Normal s1*)#*xs1*))) =
last ((*While b c*, *Normal s1*) # (*Seq c* (*While b c*), *Normal s1*) # *map* (*lift* (*While b c*)) *xs1*)
using *seq-map* **by** *fastforce*
have *throw-comm*:(Γ, (*Throw*, *Normal sl*) # *ys*) ∈ *comm*(*G*,(*p*∩(−*b*),*a*)) *F*
proof −
have *assum-throw*: (Γ, (*Throw*, *Normal sl*) # *ys*) ∈ *assum* (*a*,*R*)
using *sl-a a6 sub-assum-r*[*of* - (*LanguageCon.com.While b c*, *Normal s1*) #
(*LanguageCon.com.Seq c* (*LanguageCon.com.While b c*), *Normal s1*) #
map (*lift* (*LanguageCon.com.While b c*)) *xs1* (*Throw*, *Normal sl*)]
by *fastforce*
also have (Γ,(*Throw*, *Normal sl*) # *ys*) ∈ *cpn n* Γ *Throw* (*Normal sl*)
unfolding *cpn-def* **using** *a4* **by** *fastforce*
ultimately show *?thesis* **using** *Throw-sound*[*of a R G* Γ] *a10 a8*
unfolding *com-cvalidityn-def com-validityn-def* **by** *fast*
qed
have *p1*:(*LanguageCon.com.While b c*, *Normal s1*) #
(*LanguageCon.com.Seq c* (*LanguageCon.com.While b c*), *Normal s1*) #
map (*lift* (*LanguageCon.com.While b c*)) *xs1* ≠
[] ∧
(*LanguageCon.com.Throw*, *Normal sl*) # *ys* ≠ [] **by** *auto*
have *sl* ∈ *a* **using** *sl-a* **by** *fastforce*
then have *G1-ref*:(*Normal sl*, *Normal sl*) ∈ *G* **using** *a10* **by** *auto*
moreover have *snd* (*last* ((*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #

map (*lift* (*While b c*)) *xs1*)) = *Normal sl*
using *last-while1 last-while2 a3* **by** *fastforce*
moreover have *snd* (((*LanguageCon.com.Throw*, *Normal sl*) # *ys*) ! *0*) = *Normal sl*
by (*metis nth-Cons-0 snd-conv*)
ultimately have *G*:(*snd* (*last* ((*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
map (*lift* (*While b c*)) *xs1*)),
snd (((*LanguageCon.com.Throw*, *Normal sl*) # *ys*) ! *0*)) ∈ *G* **by** *auto*
have *cptn*:(Γ, ((*LanguageCon.com.While b c*, *Normal s1*) #
(*LanguageCon.com.Seq c* (*LanguageCon.com.While b c*), *Normal s1*) #
map (*lift* (*LanguageCon.com.While b c*)) *xs1*) @
(*LanguageCon.com.Throw*, *Normal sl*) # *ys*)
∈ *cptn* **using** *cptn a4 a0 a1 a3 a4 cptn-eq-cptn-mod-set cptn-mod.CptnModWhile3 s1-normal*
cptn-eq-cptn-mod-nest **by** (*metis append-Cons*)
show *?thesis* **using** *a0 comm-union*[*OF comm-while throw-comm p1 G cptn*] **by** *auto*
qed

inductive-cases *stepc-elim-cases-while-throw* [*cases set*]:
$\Gamma\vdash_c$(*While b c*, *s*) → (*Throw*, *t*)

lemma *WhileSound-aux*:
Γ $\models n_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] ⟹
Sta p R ⟹
Sta (*p* ∩ (−*b*)) *R* ⟹
Sta a R ⟹
(*n*, Γ,*x*)∈ *cptn-mod-nest-call* ⟹
∀ *s*. (*Normal s*, *Normal s*) ∈ *G* ⟹
∀ *s xs*. *x* = ((*While b c1*),*s*)#*xs* ⟶
(Γ,*x*)∈*assum*(*p*,*R*) ⟶
(Γ,*x*) ∈ *comm* (*G*,((*p* ∩ (−*b*)),*a*)) *F*
proof −
assume *a0*: Γ $\models n_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] **and**
a1: *Sta p R* **and**
a2: *Sta* (*p* ∩ (−*b*)) *R* **and**
a3: *Sta a R* **and**
a4: (*n*,Γ,*x*)∈ *cptn-mod-nest-call* **and**
a5: ∀ *s*. (*Normal s*, *Normal s*) ∈ *G*
{**fix** *xs s*
assume *while-xs*:*x*=((*While b c1*),*s*)#*xs* **and**
x-assum:(Γ,*x*)∈*assum*(*p*,*R*)
have (Γ,*x*) ∈ *comm* (*G*,((*p* ∩ (−*b*)),*a*)) *F*
using *a4 a0 while-xs x-assum*
proof (*induct arbitrary*: *xs s c1 rule*:*cptn-mod-nest-call.induct*)

case (*CptnModNestOne* Γ *C s1*) **thus** *?case*
  using *CptnModOne* **unfolding** *comm-def final-def*
  by *auto*
next
  case (*CptnModNestEnv* Γ *C s1 t1 n xsa*)
  then have *c-while*:*C* = *While b c1* **by** *fastforce*
  have (Γ, (*C*, *t1*) # *xsa*) ∈ *assum* (*p*, *R*) ⟶
        (Γ, (*C*, *t1*) # *xsa*) ∈ *comm* (*G*, *p* ∩ (−*b*), *a*) *F*
  using *CptnModNestEnv* **by** *fastforce*
  moreover have(*n*,Γ,(*C*, *s1*)#(*C*, *t1*) # *xsa*) ∈ *cptn-mod-nest-call*
  using *CptnModNestEnv*(*1*,*2*) *CptnModNestEnv.hyps*(*1*) *CptnModNestEnv.hyps*(*2*)
    using *cptn-mod-nest-call.CptnModNestEnv* **by** *blast*
  then have *cptn-mod*:(Γ,(*C*, *s1*)#(*C*, *t1*) # *xsa*) ∈ *cptn*
    using *cptn-eq-cptn-mod-nest* **by** *blast*
  then have (Γ, (*C*, *t1*) # *xsa*) ∈ *assum* (*p*, *R*)
    using *tl-of-assum-in-assum* *CptnModNestEnv*(*6*) *a1 a2 a3 a4 a5*
    by *blast*
  ultimately have (Γ, (*C*, *t1*) # *xsa*) ∈ *comm* (*G*, *p* ∩ (−*b*), *a*) *F*
    by *auto*
  also have ¬ ($\Gamma\vdash_c$((*C*,*s1*)) → ((*C*,*t1*)))
    by (*simp add*: *mod-env-not-component*)
  ultimately show *?case*
    using *cptn-mod etran-in-comm* **by** *blast*
next
  case (*CptnModNestSkip* Γ *C s1 t1 n xsa*)
  then have *C*=*While b c1* **by** *auto*
  also have (*n*,Γ, (*LanguageCon.com.Skip*, *t1*) # *xsa*) ∈ *cptn-mod-nest-call*
    using *cptn-eq-cptn-mod-set CptnModNestSkip*(*4*) **by** *fastforce*
  thus *?case* **using** *WhileNone CptnModNestSkip a1 a2 a3 a4 a5* **by** *blast*
next
  case (*CptnModNestThrow* Γ *C s1 t1 n xsa*)
  then have *C* = *While b c1* **by** *auto*
    thus *?case* **using** *stepc-elim-cases-while-throw CptnModNestThrow*(*1*)
    by *blast*
next
  case (*CptnModNestWhile1* *n* Γ *c s1 xs1 b1 xsa zs*)
  then have *b*=*b1* ∧ *c*=*c1* ∧ *s*=*Normal s1* **by** *auto*
  thus *?case*
  using *a4 a5 CptnModNestWhile1 while1*[*of n* Γ] **by** *blast*
next
  case (*CptnModNestWhile2 n* Γ *c s1 xs1 b1 xsa ys zs*)
  then have *a00*: (*n*,Γ, (*While b c*, *Normal s1*) #
      (*Seq c* (*While b c*), *Normal s1*) # *xsa*)∈*cptn-mod-nest-call*
    using *cptn-mod-nest-call.CptnModNestWhile2* **by** *fast*
  then have *eqs*:*b*=*b1* ∧ *c*=*c1* ∧ *s*=*Normal s1* **using** *CptnModNestWhile2* **by**
auto
  thus *?case* **using** *a00 a4 a5 CptnModNestWhile2 while2*[*of n* Γ *b c s1 xsa xs1*
ys F p R G a]
    by *blast*

next
case (*CptnModNestWhile3 n* Γ *c s1 xs1 b1 sl ys zs*)
then have *eqs*:*b*=*b1* ∧ *c*=*c1* ∧ *s*=*Normal s1* **by** *auto*
then have (Γ, (*While b c*, *Normal s1*) #
(*Seq c* (*While b c*), *Normal s1*) #
((*map* (*lift* (*While b c*)) *xs1* @
(*Throw*, *Normal sl*) # *ys*))) ∈ *comm* (*G*, *p*∩(−*b*), *a*) *F*
using *a1 a3 a4 a5 CptnModNestWhile3 while3*[*of n* Γ *c s1 xs1 b sl ys F p R G a*]
by *fastforce*
thus *?case* **using** *eqs CptnModNestWhile3* **by** *auto*
qed (*auto*)
}
then show *?thesis* **by** *auto*
qed

lemma *While-sound*:
$\Gamma,\Theta \vdash_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] ⟹
(∀ *n*. $\Gamma,\Theta \models n_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*]) ⟹
Sta p R ⟹
Sta (*p* ∩ (−*b*)) *R* ⟹ *Sta a R* ⟹ ∀ *s*. (*Normal s*, *Normal s*) ∈ *G* ⟹
$\Gamma,\Theta \models n_{/F}$ (*While b c1*) *sat* [*p*, *R*, *G*, *p* ∩ (−*b*),*a*]
proof −
assume
a0:$\Gamma,\Theta \vdash_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] **and**
a1:∀ *n*. $\Gamma,\Theta \models n_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*] **and**
a2: *Sta p R* **and**
a3: *Sta* (*p* ∩ (−*b*)) *R* **and**
a4: *Sta a R* **and**
a5: ∀ *s*. (*Normal s*, *Normal s*) ∈ *G*
{
fix *s*
assume *all-call*:∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. $\Gamma \models n_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]
then have *a1*:($\Gamma \models n_{/F}$ *c1 sat* [*p* ∩ *b*, *R*, *G*, *p*,*a*])
using *a1 com-cvalidityn-def* **by** *fastforce*
have *cpn n* Γ (*While b c1*) *s* ∩ *assum*(*p*, *R*) ⊆ *comm*(*G*, (*p* ∩ (−*b*),*a*)) *F*
proof−
{fix *c*
assume *a10*:*c* ∈ *cpn n* Γ (*While b c1*) *s* **and** *a11*:*c* ∈ *assum*(*p*, *R*)
then have *a10′*: *c* ∈ *cp* Γ (*While b c1*) *s*
unfolding *cp-def cpn-def* **using** *cptn-eq-cptn-mod-set cptn-mod-nest-cptn-mod* **by** *fastforce*
obtain Γ*1 l* **where** *c-prod*:*c*=(Γ*1*,*l*) **by** *fastforce*
have *cp*:*l*!*0*=((*While b c1*),*s*) ∧ (Γ,*l*) ∈ *cptn* ∧ Γ=Γ*1* **using** *a10′ cp-def c-prod* **by** *fastforce*
have Γ*1*:(Γ, *l*) = *c* **using** *c-prod cp* **by** *blast*
obtain *xs* **where** *l*=((*While b c1*),*s*)#*xs* **using** *cp*

proof −
  assume *a1*: $\bigwedge xs.\ l = (LanguageCon.com.While\ b\ c1,\ s)\ \#\ xs \Longrightarrow thesis$
  have $[] \neq l$
    using *cp cptn.simps* **by** *auto*
  then show *?thesis*
    using *a1* **by** (*metis* (*full-types*) *SmallStepCon.nth-tl cp*)
qed
moreover have $(n,\Gamma,l) \in$ *cptn-mod-nest-call* **using** *a10*
  using $\Gamma 1$ *cpn-def* **by** *fastforce*
ultimately have $c \in comm(G,\ (p \cap (-b),a))\ F$
using *a1 a2 a3 a4   WhileSound-aux a11* $\Gamma 1$ *a5*
  by *blast*
} **thus** *?thesis* **by** *auto* **qed**
}
thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Conseq-sound*:
$(\forall s \in p.$
  $\exists p'\ R'\ G'\ q'\ a'\ I'\ \Theta'.$
    $s \in p' \wedge$
    $R \subseteq R' \wedge$
    $G' \subseteq G \wedge$
    $q' \subseteq q \wedge$
    $a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$
    $\Gamma,\Theta' \vdash_{/F} P\ sat\ [p',R',\ G',\ q',a'] \wedge$
    $(\forall n.\ \Gamma,\Theta' \models n_{/F}\ P\ sat\ [p',\ R',\ G',\ q',a'])) \Longrightarrow$
$\Gamma,\Theta \models n_{/F}\ P\ sat\ [p,R,\ G,\ q,a]$
proof −
assume
a0: $(\forall s \in p.$
  $\exists p'\ R'\ G'\ q'\ a'\ I'\ \Theta'.$
    $s \in p' \wedge$
    $R \subseteq R' \wedge$
    $G' \subseteq G \wedge$
    $q' \subseteq q \wedge$
    $a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$
    $\Gamma,\Theta' \vdash_{/F} P\ sat\ [p',R',\ G',\ q',a'] \wedge$
    $(\forall n.\ \Gamma,\Theta' \models n_{/F}\ P\ sat\ [p',\ R',\ G',\ q',a']))$
{
  fix *s*
  assume *all-call*: $\forall (c,p,R,G,q,a) \in \Theta.\ \Gamma \models n_{/F}\ (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]$
  have $cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
  proof −
  {
    fix *c*
    assume *a10*: $c \in cpn\ n\ \Gamma\ P\ s$ **and** *a11*: $c \in assum(p,\ R)$

then have *a10′*:$c \in cp\ \Gamma\ P\ s$ **unfolding** *cp-def cpn-def cptn-eq-cptn-mod-nest* **by** *auto*

obtain $\Gamma 1\ l$ **where** *c-prod*:$c=(\Gamma 1,l)$ **by** *fastforce*

have *cp*:$l!0=(P,s) \wedge (n,\Gamma,l) \in$ *cptn-mod-nest-call* $\wedge \Gamma=\Gamma 1$ **using** *a10 cpn-def c-prod* **by** *fastforce*

have $\Gamma 1$:$(\Gamma,\ l) = c$ **using** *c-prod cp* **by** *blast*

obtain *xs* **where** $l=(P,s)\#xs$ **using** *cp*

proof −

  assume *a1*: $\bigwedge xs.\ l = (P,\ s)\ \#\ xs \Longrightarrow thesis$

  have $[] \neq l$

    using *cp cptn.simps*

    using *CptnEmpty* **by** *force*

  then show *?thesis*

    using *a1* **by** (*metis* (*full-types*) *SmallStepCon.nth-tl cp*)

qed

obtain *ns* **where** *s*:$(s = Normal\ ns)$ **using** *a10 a11* **unfolding** *assum-def cpn-def* **by** *fastforce*

then have $ns \in p$ **using** *a10 a11* **unfolding** *assum-def cpn-def* **by** *fastforce*

then have *ns*:$ns \in p$ **by** *auto*

then have

$\forall s.\ s \in p \longrightarrow (\exists p'\ R'\ G'\ q'\ a'\ \Theta'.\ (s \in p') \wedge$

  $R \subseteq R' \wedge$

  $G' \subseteq G \wedge$

  $q' \subseteq q \wedge$

  $a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$

  $(\Gamma,\Theta' \vdash_{/F} P\ sat\ [p',R',\ G',\ q',a']) \wedge$

  $(\forall n.\ \Gamma,\Theta' \models n_{/F}\ P\ sat\ [p',\ R',\ G',\ q',a']))$ **using** *a0* **by** *auto*

then have

$ns \in p \longrightarrow (\exists p'\ R'\ G'\ q'\ a'\ \Theta'.\ (ns \in p'\ ) \wedge$

  $R \subseteq R' \wedge$

  $G' \subseteq G \wedge$

  $q' \subseteq q \wedge$

  $a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$

  $(\Gamma,\Theta' \vdash_{/F} P\ sat\ [p',R',\ G',\ q',a']) \wedge$

  $(\forall n.\ \Gamma,\Theta' \models n_{/F}\ P\ sat\ [p',\ R',\ G',\ q',a']))$ **apply** (*rule allE*) **by** *auto*

then obtain $p'\ R'\ G'\ q'\ a'\ \Theta'$ **where**

rels:

$ns \in p' \wedge$

  $R \subseteq R' \wedge$

  $G' \subseteq G \wedge$

  $q' \subseteq q \wedge$

  $a' \subseteq a \wedge \Theta' \subseteq \Theta \wedge$

  $(\forall n.\ \Gamma,\Theta' \models n_{/F}\ P\ sat\ [p',\ R',\ G',\ q',a'])$ **using** *ns* **by** *auto*

then have $s \in$ *Normal* ‘ p' **using** *s* **by** *fastforce*

then have $(\Gamma,l) \in assum(p',\ R')$

  using *a11 rels cp a11 c-prod assum-R-R′*[*of* $\Gamma\ l\ p\ R\ p'\ R'$]

  by *fastforce*

then have $(\Gamma,l) \in comm(G',(q',a'))\ F$

using *rels all-call a10 c-prod cp* **unfolding** *com-cvalidityn-def com-validityn-def*

by *blast*
then have $(\Gamma,l) \in comm(G, (q,a))\ F$
using *c-prod cp comm-conseq*[*of* $\Gamma\ l\ G'\ q'\ a'\ F\ G\ q\ a$] *rels* **by** *fastforce*
then have $c \in comm(G, (q,a))\ F$ **using** *c-prod cp* **by** *fastforce*
}
thus *?thesis* **unfolding** *comm-def* **by** *force* **qed**
} thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *Conj-post-sound*:
$\Gamma,\Theta \vdash_{/F} P\ sat\ [p,R,\ G,\ q,a] \wedge$
$(\forall n.\ \Gamma,\Theta \models n_{/F}\ P\ sat\ [p,\ R,\ G,\ q,a]) \Longrightarrow$
$\Gamma,\Theta \vdash_{/F} P\ sat\ [p,R,\ G,\ q',a'] \wedge$
$(\forall n.\ \Gamma,\Theta \models n\ _{/F}\ P\ sat\ [p,\ R,\ G,\ q',a']) \Longrightarrow$
$\Gamma,\Theta \models n_{/F}\ P\ sat\ [p,R,\ G,\ q \cap q'\ ,a \cap a']$
proof −
assume *a0*: $\Gamma,\Theta \vdash_{/F} P\ sat\ [p,R,\ G,\ q,a] \wedge$
$(\forall n.\ \Gamma,\Theta \models n_{/F}\ P\ sat\ [p,\ R,\ G,\ q,a])$ **and**
a1: $\Gamma,\Theta \vdash_{/F} P\ sat\ [p,R,\ G,\ q',a'] \wedge$
$(\forall n.\ \Gamma,\Theta \models n\ _{/F}\ P\ sat\ [p,\ R,\ G,\ q',a'])$
{
fix s
assume *all-call*:$\forall (c,p,R,G,q,a) \in \Theta.\ \Gamma \models n_{/F}\ (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]$
with *a0* **have** *a0*:$cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q,a))\ F$
unfolding *com-cvalidityn-def com-validityn-def* **by** *auto*
with *a1 all-call* **have** *a1*:$cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q',a'))\ F$
unfolding *com-cvalidityn-def com-validityn-def* **by** *auto*
have $cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q\cap q',a\cap a'))\ F$
proof −
{
fix c
assume *a10*:$c \in cpn\ n\ \Gamma\ P\ s$ **and** *a11*:$c \in assum(p,\ R)$
then have $c \in comm(G,(q,a))\ F \wedge c \in comm(G,(q',a'))\ F$
using *a0 a1* **by** *auto*
then have $c \in comm(G,\ (q\cap q',a\cap a'))\ F$
unfolding *comm-def* **by** *fastforce*
}
thus *?thesis* **unfolding** *comm-def* **by** *force* **qed**
} thus *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *x91*:$sa \neq \{\} \Longrightarrow c \in comm(G,\ (\bigcap i \in sa.\ q\ i,a))\ F\ = (\forall i \in sa.\ c \in comm(G,\ q\ i,a)\ F)$
unfolding *comm-def* **apply** (*auto simp add*: *Ball-def*)
apply (*frule spec* , *force*)
by (*frule spec*, *force*)

lemma *conj-inter-sound*:
$sa \neq \{\} \Longrightarrow$
$\forall i{\in}sa.\ \Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q\ i,a] \wedge (\forall n.\ \Gamma,\Theta \models n_{/F}\ P\ sat\ [p,R,\ G,\ q\ i,a])$
$\Longrightarrow$
$\Gamma,\Theta \models n_{/F}\ P\ sat\ [p,R,\ G,\ \bigcap i{\in}sa.\ q\ i,a]$
proof –
  assume $a0'$:$sa{\neq}\{\}$ **and**
    $a0$: $\forall i{\in}sa.\ \Gamma,\Theta \vdash_{/F} P\ sat\ [p,\ R,\ G,\ q\ i,a] \wedge$
      $(\forall n.\ \Gamma,\Theta \models n_{/F}\ P\ sat\ [p,R,\ G,\ q\ i,a])$
{
  fix s
  assume *all-call*:$\forall (c,p,R,G,q,a){\in}\ \Theta.\ \Gamma \models n_{/F}\ (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]$
  with *a0* **have** *a0*:$\forall i{\in}sa.\ cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (q\ i,a))\ F$
    unfolding *com-cvalidityn-def com-validityn-def* **by** *auto*
  have $cpn\ n\ \Gamma\ P\ \ s \cap assum(p,\ R) \subseteq comm(G,\ (\bigcap i{\in}sa.\ q\ i,a))\ F$
  proof –
  {
    fix c
    assume *a10*:$c \in cpn\ n\ \Gamma\ P\ s$ **and** *a11*:$c \in assum(p,\ R)$
    then have $(\forall i{\in}sa.\ c{\in}comm(G,\ q\ i,a)\ F)$
      using *a0* **by** *fastforce*
    then have $c{\in}comm(G,\ (\bigcap i{\in}sa.\ q\ i,a))\ F$ **using** *x91*[*OF a0'*] **by** *blast*
  }
  thus *?thesis* **unfolding** *comm-def* **by** *force* **qed**
} **thus** *?thesis* **by** (*simp add*: *com-validityn-def*[*of* Γ] *com-cvalidityn-def*)
qed

lemma *localRG-sound*: $\Gamma,\Theta \vdash_{/F} c\ sat\ [p,\ R,\ G,\ q,a] \Longrightarrow (\bigwedge n.\ \Gamma,\Theta \models n\ _{/F}\ c\ sat\ [p,\ R,\ G,\ q,a])$
proof (*induct rule*:*lrghoare.induct*)
  case *Skip*
    thus *?case* **by** (*simp add*: *Skip-sound*)
next
  case *Spec*
    thus *?case* **by** (*simp add*: *Spec-sound*)
next
  case *Basic*
    thus *?case* **by** (*simp add*: *Basic-sound*)
next
  case *Await*
    thus *?case* **by** (*simp add*: *Await-sound*)
next
  case *Throw* **thus** *?case* **by** (*simp add*: *Throw-sound*)
next
  case *If* **thus** *?case* **using** *If-sound* **by** (*simp add*: *If-sound*)

next
  case *Asm* **thus** *?case* **by** (*simp add*: *Asm-sound*)
next
  case *CallRec* **thus** *?case* **by** (*simp add*: *CallRec-sound*)
next
  case *Call* **thus** *?case* **using** *Call-sound* **by** (*simp add*: *Call-sound*)
next
  case *Seq* **thus** *?case* **by** (*simp add*: *Seq-sound*)
next
  case *Catch* **thus** *?case* **by** (*simp add*: *Catch-sound*)
next
  case *DynCom* **thus** *?case* **by** (*simp add*: *DynCom-sound*)
next
  case *Guard* **thus** *?case* **by** (*simp add*: *Guard-sound*)
next
  case *Guarantee* **thus** *?case* **by** (*simp add*: *Guarantee-sound*)
next
  case *While* **thus** *?case* **by** (*simp add*: *While-sound*)
next
  case (*Conseq* p R G q a Γ Θ F P) **thus** *?case*
    using *Conseq-sound* **by** *simp*
next
  case (*Conj-post* Γ Θ F P p' R' G' q a q' a') **thus** *?case*
    using *Conj-post-sound*[*of* Γ Θ] **by** *simp*
next
  case (*Conj-Inter sa* Γ Θ F P p' R' G' q a)
    thus *?case* **using** *conj-inter-sound*[*of sa* Γ Θ] **by** *simp*
qed

definition *ParallelCom* :: $('s,'p,'f,'e)$ *rgformula list* $\Rightarrow$ $('s,'p,'f,'e)$ *par-com*
where
ParallelCom Ps $\equiv$ *map fst Ps*

lemma *ParallelCom-Com*:$i<$*length xs* $\Longrightarrow$ (*ParallelCom xs*)!i = *Com* (*xs*!i)
unfolding *ParallelCom-def Com-def* **by** *fastforce*

lemma *etran-ctran-eq-p-normal-s*: $\Gamma\vdash_c s1 \ \rightarrow s1' \Longrightarrow$
            $\Gamma\vdash_c s1 \ \rightarrow_e s1' \Longrightarrow$
            fst s1 = *fst s1'* $\wedge$ *snd s1* = *snd s1'* $\wedge$ ($\exists$ *ns1*. *snd s1* = *Normal ns1*)
proof –
   assume *a0*: $\Gamma\vdash_c s1 \ \rightarrow s1'$ **and**
          a1: $\Gamma\vdash_c s1 \ \rightarrow_e s1'$
   then obtain *ps1 ss1 ps1' ss1'* **where** *prod*:*s1* = (*ps1*,*ss1*) $\wedge$ *s1'* = (*ps1'*, *ss1'*)
     by *fastforce*
   then have *ps1*=*ps1'* **using** *a1 etranE* **by** *fastforce*
   thus *?thesis* **using** *prod a0* **by** (*simp add*: *mod-env-not-component*)
qed

lemma *step-e-step-c-eq*:⟦
$(\Gamma,l) \propto clist$;
$Suc\ m < length\ l$;
$i < length\ clist$;
$(fst\ (clist!i))\vdash_c((snd\ (clist!i))!m) \rightarrow_e\ ((snd\ (clist!i))!Suc\ m)$;
$(fst\ (clist!i))\vdash_c((snd\ (clist!i))!m) \rightarrow\ ((snd\ (clist!i))!Suc\ m)$;
$(\forall\, l{<}length\ clist.$
$l{\neq}i \longrightarrow\ (fst\ (clist!l))\vdash_c\ (snd\ (clist!l))!m\ \rightarrow_e\ ((snd\ (clist!l))!(Suc\ m)))$
⟧ $\Longrightarrow$
$l!m = l!(Suc\ m) \wedge (\exists\, ns.\ snd\ (l!m) = Normal\ ns\ )$
proof −
assume *a0*:$(\Gamma,l) \propto clist$ **and**
a1:$Suc\ m < length\ l$ **and**
a2:$i < length\ clist$ **and**
a3:$(fst\ (clist!i))\vdash_c((snd\ (clist!i))!m) \rightarrow_e\ ((snd\ (clist!i))!Suc\ m)$ **and**
a4:$(fst\ (clist!i))\vdash_c((snd\ (clist!i))!m) \rightarrow\ ((snd\ (clist!i))!Suc\ m)$ **and**
a5:$(\forall\, l{<}length\ clist.$
$l{\neq}i \longrightarrow (fst\ (clist!l))\vdash_c\ (snd\ (clist!l))!m\ \rightarrow_e\ ((snd\ (clist!l))!(Suc\ m)))$
obtain *fp fs sp ss*
where *prod-step*:
$\Gamma\vdash_c\ (fp,\ fs) \rightarrow (sp,ss) \wedge$
$fp = fst\ (((snd\ (clist!i))!m)) \wedge fs = snd\ (((snd\ (clist!i))!m)) \wedge$
$sp = fst\ ((snd\ (clist!i))!(Suc\ m)) \wedge ss = snd((snd\ (clist!i))!(Suc\ m)) \wedge$
$\Gamma = fst\ (clist!i)$
using *a0 a2 a1 a4* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
have *snd-lj*:$(snd\ (l!m)) = snd\ ((snd\ (clist!i))!m)$
using *a0 a1 a2* **unfolding** *conjoin-def same-state-def*
by *fastforce*
have *fst-clist-*Γ:$\forall\, i{<}length\ clist.\ fst(clist!i) = \Gamma$
using *a0* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
have *all-env*: $\forall\, l{<}length\ clist.$
$(fst\ (clist!l))\vdash_c\ (snd\ (clist!l))!m\ \rightarrow_e\ ((snd\ (clist!l))!(Suc\ m))$
using *a3 a5 a2 fst-clist-*Γ **by** *fastforce*
then have *allP*:$\forall\, l{<}\ length\ clist.\ fst\ ((snd\ (clist!l))!m) = fst\ ((snd\ (clist!l))!(Suc\ m))$
by (*fastforce elim*:*etranE*)
then have $fst\ (l!m) = (fst\ (l!(Suc\ m)))$
using *a0 conjoin-same-program-i-j* [*of* (Γ,*l*)] *a1* **by** *fastforce*
also have *snd-l-normal*:$snd\ (l!m) = snd\ (l!(Suc\ m)) \wedge (\exists\, ns.\ snd\ (l!m) = Normal\ ns\ )$
proof −
have $(snd\ (l!Suc\ m)) = snd\ ((snd\ (clist!i))!(Suc\ m))$
using *a0 a1 a2* **unfolding** *conjoin-def same-state-def*
by *fastforce*
also have $fs = ss \wedge$
$(\exists\, ns.\ (snd\ ((snd\ (clist!i))!m) = Normal\ ns\ ))$
using *a1 a2 all-env prod-step allP*

by (*metis step-change-p-or-eq-s*)
ultimately show *?thesis* **using** *snd-lj prod-step a1* **by** *fastforce*
qed
ultimately show *?thesis* **using** *prod-eq-iff* **by** *blast*
qed

lemma *two'*:
$[\![\forall i<length\ xs.\ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i));$
$p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)));$
$\forall i<length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)];$
$length\ xs=length\ clist;\ (\Gamma,l) \in par\text{-}cp\ \Gamma\ (ParallelCom\ xs)\ s;\ (\Gamma,l)\in par\text{-}assum\ (p,\ R)\ ;$
$\forall i<length\ clist.\ clist!i\in cp\ \Gamma\ (Com(xs!i))\ s;\ (\Gamma,l) \propto clist;(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]);$
$snd\ (last\ l) \notin Fault\ `\ F]\!]$
$\Longrightarrow \forall j\ i\ ns\ ns'.\ i<length\ clist \wedge Suc\ j<length\ l \longrightarrow$
$\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j) \longrightarrow$
$(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in Rely(xs!i)$
proof −
assume $a0{:}\forall i<length\ xs.\ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i))$ **and**
$a1{:}p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)))$ **and**
$a2{:}\forall i<length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)]$ **and**
$a3$: $length\ xs=length\ clist$ **and**
$a4$: $(\Gamma,l) \in par\text{-}cp\ \Gamma\ (ParallelCom\ xs)\ s$ **and**
$a5$: $(\Gamma,l)\in par\text{-}assum\ (p,\ R)$ **and**
$a6$: $\forall i<length\ clist.\ clist!i\in cp\ \Gamma\ (Com(xs!i))\ s$ **and**
$a7$: $(\Gamma,l) \propto clist$ **and**
$a8$: $(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a])$ **and**
$a9$: $snd\ (last\ l) \notin Fault\ `\ F$
{
assume $a10{:}\exists i\ j\ ns\ ns'.$
$i<length\ clist \wedge Suc\ j<length\ l \wedge$
$\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j) \wedge$
$\neg(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in Rely(xs!i)$
then obtain j **where**
$a10{:}\exists i\ ns\ ns'.$
$i<length\ clist \wedge Suc\ j<length\ l \wedge$
$\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j) \wedge$
$\neg(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in Rely(xs!i)$ **by** *fastforce*
let $?P = \lambda j.\ \exists i.\ i<length\ clist \wedge Suc\ j<length\ l \wedge$
$\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j) \wedge$
$(\neg\ (snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in Rely(xs!i))$
obtain m **where** *fist-occ*:$(?P\ m) \wedge (\forall i<m.\ \neg\ ?P\ i)$ **using** *exists-first-occ*[*of ?P*

j] *a10* **by** *blast*
then have *?P m* **by** *fastforce*
then obtain *i* **where**
fst-occ:$i<$*length clist* $\wedge$ *Suc m*$<$*length l* $\wedge$
$\Gamma\vdash_c$((*snd* (*clist*!*i*))!*m*) $\rightarrow_e$ ((*snd* (*clist*!*i*))!*Suc m*) $\wedge$
($\neg$ (*snd*((*snd* (*clist*!*i*))!*m*), *snd*((*snd* (*clist*!*i*))!*Suc m*)) $\in$ *Rely*(*xs*!*i*))
by *fastforce*
have *notP*:($\forall$ *i*$<$*m*. $\neg$ *?P i*) **using** *fist-occ* **by** *blast*
have *fst-clist-*Γ:$\forall$ *i*$<$*length clist*. *fst*(*clist*!*i*) = Γ
using *a7* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
have *compat*:($\Gamma\vdash_p$(*l*!*m*) $\rightarrow$ (*l*!(*Suc m*))) $\wedge$
($\exists$ *i*$<$*length clist*.
((*fst* (*clist*!*i*))$\vdash_c$ ((*snd* (*clist*!*i*))!*m*) $\rightarrow$ ((*snd* (*clist*!*i*))!(*Suc m*))) $\wedge$
($\forall$ *l*$<$*length clist*.
l$\neq$*i* $\longrightarrow$ (*fst* (*clist*!*l*))$\vdash_c$ (*snd* (*clist*!*l*))!*m* $\rightarrow_e$ ((*snd* (*clist*!*l*))!(*Suc m*)))) $\vee$
($\Gamma\vdash_p$(*l*!*m*) $\rightarrow_e$ (*l*!(*Suc m*)) $\wedge$
($\forall$ *i*$<$*length clist*. (*fst* (*clist*!*i*))$\vdash_c$ (*snd* (*clist*!*i*))!*m* $\rightarrow_e$ ((*snd* (*clist*!*i*))!(*Suc m*))))
using *a7 fst-occ* **unfolding** *conjoin-def compat-label-def* **by** *simp*
{
assume *a20*: ($\Gamma\vdash_p$(*l*!*m*) $\rightarrow_e$ (*l*!(*Suc m*)) $\wedge$
($\forall$ *i*$<$*length clist*. (*fst* (*clist*!*i*))$\vdash_c$ (*snd* (*clist*!*i*))!*m* $\rightarrow_e$ ((*snd* (*clist*!*i*))!(*Suc m*))))
then have (*snd* (*l*!*m*),*snd* (*l*!(*Suc m*))) $\in$ *R*
using *fst-occ a5* **unfolding** *par-assum-def* **by** *fastforce*
then have (*snd*(*l*!*m*), *snd*(*l*!(*Suc m*))) $\in$ *Rely*(*xs*!*i*)
using *fst-occ a3 a0* **by** *fastforce*
then have (*snd* ((*snd* (*clist*!*i*))!*m*), *snd* ((*snd* (*clist*!*i*))!(*Suc m*))) $\in$ *Rely*(*xs*!*i*)
using *a7 fst-occ* **unfolding** *conjoin-def same-state-def* **by** *fastforce*
then have *False* **using** *fst-occ* **by** *auto*
}note *l* = *this*
{
assume *a20*:($\Gamma\vdash_p$(*l*!*m*) $\rightarrow$ (*l*!(*Suc m*))) $\wedge$
($\exists$ *i*$<$*length clist*.
((*fst* (*clist*!*i*))$\vdash_c$ ((*snd* (*clist*!*i*))!*m*) $\rightarrow$ ((*snd* (*clist*!*i*))!(*Suc m*))) $\wedge$
($\forall$ *l*$<$*length clist*.
l$\neq$*i* $\longrightarrow$ (*fst* (*clist*!*l*))$\vdash_c$ (*snd* (*clist*!*l*))!*m* $\rightarrow_e$ ((*snd* (*clist*!*l*))!(*Suc m*))))
then obtain *i'*
where *i'*:*i'*$<$*length clist* $\wedge$
((*fst* (*clist*!*i'*))$\vdash_c$ ((*snd* (*clist*!*i'*))!*m*) $\rightarrow$ ((*snd* (*clist*!*i'*))!(*Suc m*))) $\wedge$
($\forall$ *l*$<$*length clist*.
l$\neq$*i'* $\longrightarrow$ (*fst* (*clist*!*l*))$\vdash_c$ (*snd* (*clist*!*l*))!*m* $\rightarrow_e$ ((*snd* (*clist*!*l*))!(*Suc m*)))
by *fastforce*
then have *eq-*Γ:Γ = *fst* (*clist*!*i'*) **using** *a7* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*

obtain *fp fs sp ss*
where *prod-step*:
$\Gamma\vdash_c (fp, fs) \rightarrow (sp,ss) \wedge$
$fp = fst\ (((snd\ (clist!i'))!m)) \wedge fs = snd\ (((snd\ (clist!i'))!m)) \wedge$
$sp = fst\ ((snd\ (clist!i'))!(Suc\ m)) \wedge ss = snd((snd\ (clist!i'))!(Suc\ m))$
$\wedge$
$\Gamma = fst\ (clist!i')$
using *a7* i' **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
then have *False*
proof $(cases\ i = i')$
case *True*
then have $l!m = l!(Suc\ m) \wedge (\exists\, ns.\ snd\ (l!m) = Normal\ ns\ )$
using *step-e-step-c-eq*[*OF a7*] i' *fst-occ eq-*Γ **by** *blast*
then have $\Gamma\vdash_p(l!m)\ \rightarrow_e (l!(Suc\ m))$
using *step-pe.ParEnv* **by** (*metis prod.collapse*)
then have $(snd\ (l\ !\ m),\ snd\ (l\ !\ Suc\ m)) \in R$
using *fst-occ a5* **unfolding** *par-assum-def* **by** *fastforce*
then have $(snd\ (l\ !\ m),\ snd\ (l\ !\ Suc\ m)) \in Rely\ (xs\ !\ i)$
using *a0 a3 fst-occ* **by** *fastforce*
then show *?thesis* **using** *fst-occ a7*
unfolding *conjoin-def same-state-def*
by *fastforce*
next
case *False* **note** *not-eq = this*
thus *?thesis*
proof $(cases\ fp = sp)$
case *True*
then have $fs = ss \wedge (\exists\, ns.\ fs{=}Normal\ ns)$
using *prod-step prod-step*
using *step-change-p-or-eq-s* **by** *blast*

then have $\Gamma\vdash_c (fp,\ fs) \rightarrow_e (sp,\ ss)$ **using** *True step-e.Env*
by *fastforce*
then have $l!m = l!(Suc\ m) \wedge (\exists\, ns.\ snd\ (l!m) = Normal\ ns\ )$
using *step-e-step-c-eq*[*OF a7*] *prod-step* i' *fst-occ prod.collapse* **by** *auto*
then have $\Gamma\vdash_p(l!m)\ \rightarrow_e (l!(Suc\ m))$
using *step-pe.ParEnv* **by** (*metis prod.collapse*)
then have $(snd\ (l\ !\ m),\ snd\ (l\ !\ Suc\ m)) \in R$
using *fst-occ a5* **unfolding** *par-assum-def* **by** *fastforce*
then have $(snd\ (l\ !\ m),\ snd\ (l\ !\ Suc\ m)) \in Rely\ (xs\ !\ i)$
using *a0 a3 fst-occ* **by** *fastforce*
then show *?thesis* **using** *fst-occ a7*
unfolding *conjoin-def same-state-def*
by *fastforce*
next
case *False*
let $?l1 = take\ (Suc\ (Suc\ m))\ (snd(clist!i'))$
have *clist-cptn*:$(\Gamma,snd(clist!i')) \in cptn$ **using** *a6* i' **unfolding** *cp-def* **by**
fastforce

have *sucm-len*:*Suc m* < *length* (*snd* (*clist*!*i*′))
using *i*′ *fst-occ a7* **unfolding** *conjoin-def same-length-def* **by** *fastforce*

then have *summ-lentake*:*Suc m* < *length ?l1* **by** *fastforce*
have *len-l*: *0*<*length l* **using** *fst-occ* **by** *fastforce*
also then have *snd* (*clist*!*i*′)≠[]
using *i*′ *a7* **unfolding** *conjoin-def same-length-def* **by** *fastforce*
ultimately have *snd* (*last* (*snd* (*clist* ! *i*′))) = *snd* (*last l*)
using *a7 i*′ *conjoin-last-same-state* **by** *fastforce*
then have *last-i-notF*:*snd* (*last* (*snd*(*clist*!*i*′))) ∉ *Fault* ‘ *F*
using *a9* **by** *auto*
have ∀ *i*<*length* (*snd*(*clist*!*i*′)). *snd* (*snd*(*clist*!*i*′) ! *i*) ∉ *Fault* ‘ *F*
using *last-not-F*[*OF clist-cptn last-i-notF*] **by** *auto*
also have *suc-m-i*′:*Suc m* < *length* (*snd* (*clist* !*i*′))
using *fst-occ i*′ *a7* **unfolding** *conjoin-def same-length-def* **by** *fastforce*
ultimately have *last-take-not-f*:*snd* (*last* (*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′))))
∉ *Fault* ‘ *F*
by (*simp add*: *take-Suc-conv-app-nth*)
have *not-env-step*:¬ Γ$\vdash_c$ *snd* (*clist* ! *i*′) ! *m* $\rightarrow_e$ *snd* (*clist* ! *i*′) ! *Suc m*
using *False etran-ctran-eq-p-normal-s i*′ *prod-step* **by** *blast*
then have *snd* ((*snd*(*clist*!*i*′))!*0*)∈ *Normal* ‘ *p*
using *len-l a7 i*′ *a5* **unfolding** *conjoin-def same-state-def par-assum-def*
by *fastforce*
then have *snd* ((*snd*(*clist*!*i*′))!*0*)∈ *Normal* ‘ (*Pre* (*xs* ! *i*′))
using *a1 i*′ *a3* **by** *fastforce*
then have *snd* ((*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′)))!*0*)∈ *Normal* ‘(*Pre*
(*xs* ! *i*′))
by *fastforce*
moreover have
∀ *j*. *Suc j* < *Suc* (*Suc m*) ⟶
Γ$\vdash_c$ *snd* (*clist* ! *i*′) ! *j* $\rightarrow_e$ *snd* (*clist* ! *i*′) ! *Suc j* ⟶
(*snd* (*snd* (*clist* ! *i*′) ! *j*), *snd* (*snd* (*clist* ! *i*′) ! *Suc j*)) ∈ *Rely* (*xs* !
i′)
using *not-env-step fst-occ Suc-less-eq fist-occ i*′ *less-SucE less-trans-Suc*
by *auto*
then have ∀ *j*. *Suc j* < *length* (*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′))) ⟶
Γ$\vdash_c$ *snd* (*clist* ! *i*′) ! *j* $\rightarrow_e$ *snd* (*clist* ! *i*′) ! *Suc j* ⟶
(*snd* (*snd* (*clist* ! *i*′) ! *j*), *snd* (*snd* (*clist* ! *i*′) ! *Suc j*)) ∈ *Rely* (*xs* ! *i*′)
by *fastforce*
ultimately have (Γ, (*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′)))) ∈
assum ((*Pre* (*xs* ! *i*′)),*Rely* (*xs* ! *i*′))
unfolding *assum-def* **by** *fastforce*
moreover have (Γ,*snd*(*clist*!*i*′)) ∈ *cptn* **using** *a6 i*′ **unfolding** *cp-def* **by**
fastforce
then have (Γ,*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′))) ∈ *cptn*
by (*simp add*: *takecptn-is-cptn*)
then have (Γ,*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′))) ∈ *cp* Γ (*Com*(*xs*!*i*′)) *s*
using *i*′ *a3 a6* **unfolding** *cp-def* **by** *fastforce*
ultimately have *t*:(Γ,*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i*′))) ∈

comm (*Guar* (*xs* ! *i'*), (*Post* (*xs* ! *i'*),*Abr* (*xs* ! *i'*))) *F*
using *a8 a2 a3 i'* **unfolding** *com-cvalidity-def com-validity-def* **by** *fastforce*

have (*snd*(*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i'*))!*m*),
snd(*take* (*Suc* (*Suc m*)) (*snd*(*clist*!*i'*))!(*Suc m*))) ∈ *Guar* (*xs* ! *i'*)
using *eq*-Γ *i' comm-dest1*[*OF t last-take-not-f summ-lentake*] **by** *fastforce*

then have (*snd*((*snd*(*clist*!*i'*))!*m*),
snd((*snd*(*clist*!*i'*))!(*Suc m*))) ∈ *Guar* (*xs* ! *i'*)
by *fastforce*
then have (*snd*((*snd*(*clist*!*i*))!*m*),
snd((*snd*(*clist*!*i*))!(*Suc m*))) ∈ *Guar* (*xs* ! *i'*)
using *a7 fst-occ* **unfolding** *conjoin-def same-state-def* **by** (*metis Suc-lessD i' snd-conv*)
then have (*snd*((*snd*(*clist*!*i*))!*m*),
snd((*snd*(*clist*!*i*))!(*Suc m*))) ∈ *Rely* (*xs* ! *i*)
using *not-eq a0 i' a3 fst-occ* **by** *auto*
then have *False* **using** *fst-occ* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed
}
then have *False* **using** *compat l* **by** *auto*
} **thus** *?thesis* **by** *auto*
qed

lemma *two*:
⟦ ∀ *i*<*length xs*. *R* ∪ (⋃*j*∈{*j*. *j* < *length xs* ∧ *j* ≠ *i*}. (*Guar* (*xs* ! *j*)))
⊆ (*Rely* (*xs* ! *i*));
p ⊆ (⋂*i*<*length xs*. (*Pre* (*xs* ! *i*)));
∀ *i*<*length xs*.
Γ,Θ $\models_{/F}$ *Com* (*xs* ! *i*) *sat* [*Pre* (*xs*!*i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*),*Abr* (*xs* ! *i*)];
length xs=*length clist*; (Γ,*l*) ∈ *par-cp* Γ (*ParallelCom xs*) *s*; (Γ,*l*)∈*par-assum* (*p*, *R*);
∀ *i*<*length clist*. *clist*!*i*∈*cp* Γ (*Com*(*xs*!*i*)) *s*; (Γ,*l*) ∝ *clist*;(∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. Γ $\models_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]);
snd (*last l*) ∉ *Fault* ' *F*⟧
⟹ ∀ *j i ns ns'*. *i*<*length clist* ∧ *Suc j*<*length l* ⟶
Γ$\vdash_c$((*snd* (*clist*!*i*))!*j*) → ((*snd* (*clist*!*i*))!*Suc j*) ⟶
(*snd*((*snd* (*clist*!*i*))!*j*), *snd*((*snd* (*clist*!*i*))!*Suc j*)) ∈ *Guar*(*xs*!*i*)
proof −
assume *a0*:∀ *i*<*length xs*. *R* ∪ (⋃*j*∈{*j*. *j* < *length xs* ∧ *j* ≠ *i*}. (*Guar* (*xs* ! *j*)))
⊆ (*Rely* (*xs* ! *i*)) **and**
a1:*p* ⊆ (⋂*i*<*length xs*. (*Pre* (*xs* ! *i*))) **and**
a2:∀ *i*<*length xs*.
Γ,Θ $\models_{/F}$ *Com* (*xs* ! *i*) *sat* [*Pre* (*xs*!*i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*),*Abr* (*xs* ! *i*)] **and**

a3: *length xs=length clist* **and**
a4: $(\Gamma,l) \in$ *par-cp* Γ (*ParallelCom xs*) *s* **and**
a5: $(\Gamma,l)\in$*par-assum* (*p*, *R*) **and**
a6: $\forall$ *i<length clist*. *clist*!*i*$\in$*cp* Γ (*Com*(*xs*!*i*)) *s* **and**
a7: $(\Gamma,l) \propto$ *clist* **and**
a8: ($\forall$ (*c*,*p*,*R*,*G*,*q*,*a*)$\in$ Θ. $\Gamma \models_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*]) **and**
a9: *snd* (*last l*) $\notin$ *Fault* ' *F*

{
assume *a10*:($\exists$ *i j*. *i<length clist* $\wedge$ *Suc j<length l* $\wedge$
$\Gamma\vdash_c$((*snd* (*clist*!*i*))!*j*) $\rightarrow$ ((*snd* (*clist*!*i*))!*Suc j*) $\wedge$
$\neg$ (*snd*((*snd* (*clist*!*i*))!*j*), *snd*((*snd* (*clist*!*i*))!*Suc j*)) $\in$ *Guar*(*xs*!*i*))
then obtain *j* **where** *a10*: $\exists$ *i*. *i<length clist* $\wedge$ *Suc j<length l* $\wedge$
$\Gamma\vdash_c$((*snd* (*clist*!*i*))!*j*) $\rightarrow$ ((*snd* (*clist*!*i*))!*Suc j*) $\wedge$
$\neg$ (*snd*((*snd* (*clist*!*i*))!*j*), *snd*((*snd* (*clist*!*i*))!*Suc j*)) $\in$ *Guar*(*xs*!*i*)
by *fastforce*
let *?P* = λ*j*. $\exists$ *i*. *i<length clist* $\wedge$ *Suc j<length l* $\wedge$
$\Gamma\vdash_c$((*snd* (*clist*!*i*))!*j*) $\rightarrow$ ((*snd* (*clist*!*i*))!*Suc j*) $\wedge$
$\neg$ (*snd*((*snd* (*clist*!*i*))!*j*), *snd*((*snd* (*clist*!*i*))!*Suc j*)) $\in$ *Guar*(*xs*!*i*)
obtain *m* **where** *fist-occ*:*?P m* $\wedge$ ($\forall$ *i<m*. $\neg$ *?P i*) **using** *exists-first-occ*[*of ?P j*] *a10* **by** *blast*
then have *P*:*?P m* **by** *fastforce*
then have *notP*:($\forall$ *i<m*. $\neg$ *?P i*) **using** *fist-occ* **by** *blast*
obtain *i ns ns'* **where** *fst-occ*:*i<length clist* $\wedge$ *Suc m<length l* $\wedge$
$\Gamma\vdash_c$((*snd* (*clist*!*i*))!*m*) $\rightarrow$ ((*snd* (*clist*!*i*))!*Suc m*) $\wedge$
($\neg$ (*snd*((*snd* (*clist*!*i*))!*m*), *snd*((*snd* (*clist*!*i*))!*Suc m*)) $\in$ *Guar*(*xs*!*i*))
using *P* **by** *fastforce*
have *fst-clist-i*: *fst* (*clist*!*i*) = Γ
using *a7 fst-occ* **unfolding** *conjoin-def same-functions-def*
by *fastforce*
have *clist*!*i*$\in$*cp* Γ (*Com*(*xs*!*i*)) *s* **using** *a6 fst-occ* **by** *fastforce*
then have *clistcp*:(Γ, *snd* (*clist*!*i*))$\in$*cp* Γ (*Com*(*xs*!*i*)) *s*
using *fst-occ a7* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
let *?li=take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))
have $\Gamma \models_{/F}$ *Com* (*xs* ! *i*) *sat* [*Pre* (*xs*!*i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*),*Abr* (*xs* ! *i*)]
using *a8 a2 a3 fst-occ* **unfolding** *com-cvalidity-def* **by** *fastforce*
moreover have *take-in-ass*:(Γ, *take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))) $\in$ *assum* (*Pre*(*xs*!*i*), *Rely*(*xs*!*i*))
proof –
have *length-take-length-l*:*length* (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))) $\leq$ *length l*
using *a7 fst-occ* **unfolding** *conjoin-def same-length-def* **by** *auto*
have *snd*((*?li*!*0*)) $\in$ *Normal* ' *Pre*(*xs*!*i*)
proof –
have (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*)))!*0* = (*snd* (*clist*!*i*))!*0* **by** *fastforce*
moreover have *snd* (*snd*(*clist*!*i*)!*0*) = *snd* (*l*!*0*)
using *a7 fst-occ* **unfolding** *conjoin-def same-state-def* **by** *fastforce*
moreover have *snd* (*l*!*0*) $\in$ *Normal* ' *p*
using *a5* **unfolding** *par-assum-def* **by** *fastforce*

ultimately show *?thesis* **using** *a1 a3 fst-occ* **by** *fastforce*
qed note *left=this*
thus *?thesis*
using *two'*[*OF a0 a1 a2 a3 a4 a5 a6 a7 a8 a9*] *fst-occ* **unfolding** *assum-def* **by** *fastforce*
qed
moreover have $(\Gamma$,*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))) $\in$ *cp* Γ (*Com*(*xs*!*i*)) *s*
using *takecptn-is-cptn clistcp* **unfolding** *cp-def* **by** *fastforce*
ultimately have *comm*:(Γ, *take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*)))$\in$*comm*(*Guar*(*xs*!*i*),(*Post* (*xs* ! *i*),*Abr* (*xs* ! *i*))) *F*
unfolding *com-validity-def* **by** *fastforce*
also have *not-fault*:*snd* (*last* (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*)))) $\notin$ *Fault* ‘ *F*
proof –
have *cptn*:(Γ, *snd* (*clist*!*i*)) $\in$ *cptn*
using *fst-clist-i a6 fst-occ* **unfolding** *cp-def* **by** *fastforce*
then have (*snd* (*clist*!*i*))$\neq$[]
using *cptn.simps list.simps*(*3*)
by *fastforce*
then have *snd* (*last* (*snd* (*clist*!*i*))) = *snd* (*last l*)
using *conjoin-last-same-state fst-occ a7* **by** *fastforce*
then have *snd* (*last* (*snd* (*clist*!*i*))) $\notin$ *Fault* ‘ *F* **using** *a9*
by *simp*
also have *sucm*:*Suc m* < *length* (*snd* (*clist*!*i*))
using *fst-occ a7* **unfolding** *conjoin-def same-length-def* **by** *fastforce*
ultimately have *sucm-not-fault*:*snd* ((*snd* (*clist*!*i*))!(*Suc m*)) $\notin$ *Fault* ‘ *F*
using *last-not-F cptn* **by** *blast*
have *length* (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))) = *Suc* (*Suc m*)
using *sucm* **by** *fastforce*
then have *last* (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*))) = (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*)))!(*Suc m*)
by (*metis Suc-diff-1 Suc-inject last-conv-nth list.size*(*3*) *old.nat.distinct*(*2*) *zero-less-Suc*)
moreover have (*take* (*Suc* (*Suc m*)) (*snd* (*clist*!*i*)))!(*Suc m*) = (*snd* (*clist*!*i*))!(*Suc m*)
by *fastforce*
ultimately show *?thesis* **using** *sucm-not-fault* **by** *fastforce*
qed
then have (*Suc m* < *length* (*snd* (*clist* ! *i*))) $\longrightarrow$
($\Gamma\vdash_c$ (*snd* (*clist* ! *i*)) ! *m* $\rightarrow$ (*snd* (*clist* ! *i*)) ! *Suc m*) $\longrightarrow$
(*snd* ((*snd* (*clist* ! *i*)) ! *m*), *snd* ((*snd* (*clist* ! *i*)) ! *Suc m*)) $\in$ *Guar*(*xs*!*i*)
using *comm-dest* [*OF comm not-fault*] **by** *auto*
then have *False* **using** *fst-occ* **using** *a7* **unfolding** *conjoin-def same-length-def* **by** *fastforce*
} **thus** *?thesis* **by** *fastforce*
qed

lemma *par-cptn-env-comp*:

$(\Gamma,l) \in$ *par-cptn* $\wedge$ *Suc* $i <$ *length* $l \Longrightarrow$
$\Gamma \vdash_p l!i \rightarrow_e (l!(Suc\ i)) \vee \Gamma \vdash_p l!i \rightarrow (l!(Suc\ i))$
proof −
assume *a0*:$(\Gamma,l) \in$ *par-cptn* $\wedge$ *Suc i<length l*
then obtain *c1 s1 c2 s2* **where** *li*:$l!i=(c1,s1) \wedge l!(Suc\ i) = (c2,s2)$ **by** *fastforce*
obtain *xs ys* **where** *l*:$l= xs@((l!i)\#(l!(Suc\ i))\#ys)$ **using** *a0*
by (*metis Cons-nth-drop-Suc Suc-less-SucD id-take-nth-drop less-SucI*)
moreover then have $(drop\ (length\ xs)\ l) = ((l!i)\#(l!(Suc\ i))\#ys)$
by (*metis append-eq-conv-conj*)
moreover then have *length xs* $<$ *length l* **using** *leI* **by** *fastforce*
ultimately have $(\Gamma,((l!i)\#(l!(Suc\ i))\#ys))\in$ *par-cptn*
using *a0 droppar-cptn-is-par-cptn* **by** *fastforce*
also then have $(\Gamma,(l!(Suc\ i))\#ys)\in$ *par-cptn* **using** *par-cptn-dest li* **by** *fastforce*
ultimately show *?thesis* **using** *li par-cptn-elim-cases(2)*
by *metis*
qed

lemma *three*:
$[\![xs\neq[]; \forall i<length\ xs.\ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i));$
$p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)));$
$\forall i<length\ xs.$
$\Gamma,\Theta \models_{/F}$ *Com* $(xs\ !\ i)$ *sat* [*Pre* $(xs!i)$, *Rely* $(xs\ !\ i)$, *Guar* $(xs\ !\ i)$, *Post* $(xs\ !\ i)$,*Abr* $(xs\ !\ i)$];
length xs=length clist; $(\Gamma,l) \in$ *par-cp* Γ (*ParallelCom xs*) *s*; $(\Gamma,l) \in$ *par-assum*(p, R);
$\forall i<length\ clist.\ clist!i\in cp\ \Gamma\ (Com(xs!i))\ s$; $(\Gamma,l) \propto clist$; $(\forall (c,p,R,G,q,a)\in \Theta.$ $\Gamma \models_{/F}$ (*Call c*) *sat* $[p,\ R,\ G,\ q,a]$);
snd (*last l*) $\notin$ *Fault* ‘ $F]\!]$
$\Longrightarrow \forall j\ i.\ i<length\ clist \wedge Suc\ j<length\ l \longrightarrow\ \Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j) \longrightarrow$
$(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in$
$(R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j))))$
proof −
assume *a0*:$xs\neq[]$ **and**
a1:$\forall i<length\ xs.\ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i))$ **and**
a2: $p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)))$ **and**
a3: $\forall i<length\ xs.$
$\Gamma,\Theta \models_{/F}$ *Com* $(xs\ !\ i)$ *sat* [*Pre* $(xs!i)$, *Rely* $(xs\ !\ i)$, *Guar* $(xs\ !\ i)$, *Post* $(xs\ !\ i)$,*Abr* $(xs\ !\ i)$] **and**
a4: *length xs=length clist* **and**
a5: $(\Gamma,l) \in$ *par-cp* Γ (*ParallelCom xs*) *s* **and**
a6: $(\Gamma,l) \in$ *par-assum*(p, R) **and**
a7: $\forall i<length\ clist.\ clist!i\in cp\ \Gamma\ (Com(xs!i))\ s$ **and**
a8: $(\Gamma,l) \propto clist$ **and**
a9: $(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F}$ (*Call c*) *sat* $[p,\ R,\ G,\ q,a]$) **and**
10: *snd* (*last l*) $\notin$ *Fault* ‘ *F*

{
fix *j i ns ns'*
assume *a00*:i<*length clist* ∧ *Suc j*<*length l* **and**
a11: $\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow_e\ ((snd\ (clist!i))!Suc\ j)$
then have *two*:$\forall j\ i\ ns\ ns'$. $i<length\ clist \wedge Suc\ j<length\ l \longrightarrow$
$\Gamma\vdash_c((snd\ (clist!i))!j) \rightarrow\ ((snd\ (clist!i))!Suc\ j) \longrightarrow$
$(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in (Guar(xs!i))$
using *two*[*OF a1 a2 a3 a4 a5 a6 a7 a8 a9 10*] **by** *auto*
then have *j-lenl*:*Suc j*<*length l* **using** *a00* **by** *fastforce*
have *i-lj*:$i<length\ (fst\ (l!j)) \wedge i<length\ (fst\ (l!(Suc\ j)))$
using *conjoin-same-length a00 a8* **by** *fastforce*
have *fst-clist-Γ*:$\forall i<length\ clist.\ fst(clist!i) = \Gamma$ **using** *a8* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
have $(\Gamma\vdash_p(l!j)\ \rightarrow (l!(Suc\ j))) \wedge$
$(\exists i<length\ clist.$
$((fst\ (clist!i))\vdash_c ((snd\ (clist!i))!j)\ \rightarrow ((snd\ (clist!i))!(Suc\ j))) \wedge$
$(\forall l<length\ clist.$
$l\neq i \longrightarrow\ (fst\ (clist!l))\vdash_c (snd\ (clist!l))!j\ \rightarrow_e ((snd\ (clist!l))!(Suc\ j))))$
$\vee$
$(\Gamma\vdash_p(l!j)\ \rightarrow_e (l!(Suc\ j)) \wedge$
$(\forall i<length\ clist.\ (fst\ (clist!i))\vdash_c (snd\ (clist!i))!j\ \rightarrow_e ((snd\ (clist!i))!(Suc\ j))))$
using *a8 a00* **unfolding** *conjoin-def compat-label-def* **by** *simp*
then have *compat-label*:$(\Gamma\vdash_p(l!j)\ \rightarrow (l!(Suc\ j))) \wedge$
$(\exists i<length\ clist.$
$(\Gamma\vdash_c ((snd\ (clist!i))!j)\ \rightarrow ((snd\ (clist!i))!(Suc\ j))) \wedge$
$(\forall l<length\ clist.$
$l\neq i \longrightarrow\ \Gamma\vdash_c (snd\ (clist!l))!j\ \rightarrow_e ((snd\ (clist!l))!(Suc\ j)))) \vee$
$(\Gamma\vdash_p(l!j)\ \rightarrow_e (l!(Suc\ j)) \wedge$
$(\forall i<length\ clist.\ \Gamma\vdash_c (snd\ (clist!i))!j\ \rightarrow_e ((snd\ (clist!i))!(Suc\ j))))$
using *fst-clist-Γ* **by** *blast*
then have $(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in$
$(R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ Guar\ (xs\ !\ j)))$
proof
assume *a10*:$(\Gamma\vdash_p(l!j)\ \rightarrow (l!(Suc\ j))) \wedge$
$(\exists i<length\ clist.$
$(\Gamma\vdash_c ((snd\ (clist!i))!j)\ \rightarrow ((snd\ (clist!i))!(Suc\ j))) \wedge$
$(\forall l<length\ clist.$
$l\neq i \longrightarrow\ \Gamma\vdash_c (snd\ (clist!l))!j\ \rightarrow_e ((snd\ (clist!l))!(Suc\ j))))$
then obtain i' **where**
a20:$i'<length\ clist \wedge$
$(\Gamma\vdash_c ((snd\ (clist!i'))!j)\ \rightarrow ((snd\ (clist!i'))!(Suc\ j))) \wedge$
$(\forall l<length\ clist.$
$l\neq i' \longrightarrow\ \Gamma\vdash_c (snd\ (clist!l))!j\ \rightarrow_e ((snd\ (clist!l))!(Suc\ j)))$ **by** *blast*

thus *?thesis*
proof (*cases* $i'=i$)
case *True* **note** *eq-i* = *this*
then obtain *P S1 S2* **where** *P*:$(snd\ (clist!i'))!j=(P,S1) \wedge ((snd\ (clist!i'))!(Suc$

j)) = (*P*,*S2*)
using *a11* **by** (*fastforce elim*:*etranE*)
thus *?thesis*
proof (*cases S1* = *S2*)
case *True*
have *snd-lj*:(*snd* (*l*!*j*)) = *snd* ((*snd* (*clist*!*i'*))!*j*)
using *a8 a20 a00* **unfolding** *conjoin-def same-state-def*
by *fastforce*
have *all-e*:(∀ *l*<*length clist*. $\Gamma\vdash_c$ (*snd* (*clist*!*l*))!*j* $\rightarrow_e$ ((*snd* (*clist*!*l*))!(*Suc j*)))
using *a11 a20 eq-i* **by** *fastforce*
then have *allP*:∀ *l*< *length clist*. *fst* ((*snd* (*clist*!*l*))!*j*) = *fst* ((*snd* (*clist*!*l*))!(*Suc j*))
by (*fastforce elim*:*etranE*)
then have *fst* (*l*!*j*) = (*fst* (*l*!(*Suc j*)))
using *a8 conjoin-same-program-i-j* [*of* (Γ,*l*)] *a00* **by** *fastforce*
also have *snd* (*l*!*j*) = *snd* (*l*!(*Suc j*))
proof −
have (*snd* (*l*!*Suc j*)) = *snd* ((*snd* (*clist*!*i'*))!(*Suc j*))
using *a8 a20 a00* **unfolding** *conjoin-def same-state-def*
by *fastforce*
then show *?thesis* **using** *snd-lj P True* **by** *auto*
qed
ultimately have *l*!*j* = *l*!(*Suc j*) **by** (*simp add*: *prod-eq-iff*)
moreover have *ns1*:∃ *ns1*. *S1*=*Normal ns1*
using *P a20 step-change-p-or-eq-s* **by** *fastforce*
ultimately have $\Gamma\vdash_p$(*l*!*j*) $\rightarrow_e$ (*l*!(*Suc j*))
using *P step-pe*.*ParEnv snd-lj* **by** (*metis prod*.*collapse snd-conv*)
then have (*snd* (*l* ! *j*), *snd* (*l* ! *Suc j*)) ∈ *R*
using *a00 a6* **unfolding** *par-assum-def* **by** *fastforce*
then show *?thesis* **using** *a8 a00*
unfolding *conjoin-def same-state-def*
by *fastforce*
next
case *False* **thus** *?thesis*
using *a20 P a11 step-change-p-or-eq-s* **by** *fastforce*
qed
next
case *False*
have *i'-clist*:*i'* < *length clist* **using** *a20* **by** *fastforce*
then have *clist-i'-Guardxs*:(*snd*((*snd* (*clist*!*i'*))!*j*), *snd*((*snd* (*clist*!*i'*))!*Suc j*)) ∈ *Guar*(*xs*!*i'*)
using *two a00 False a8* **unfolding** *conjoin-def same-state-def*
by (*metis a20*)
have *snd*((*snd* (*clist*!*i*))!*j*) = *snd* (*l*!*j*) ∧ *snd*((*snd* (*clist*!*i*))!*Suc j*) = *snd* (*l*!*Suc j*)
using *a00 a20 a8* **unfolding** *conjoin-def same-state-def* **by** *fastforce*
also have *snd*((*snd* (*clist*!*i'*))!*j*) = *snd* (*l*!*j*) ∧ *snd*((*snd* (*clist*!*i'*))!*Suc j*) = *snd* (*l*!*Suc j*)

using *j-lenl a20 a8* **unfolding** *conjoin-def same-state-def* **by** *fastforce*
ultimately have $snd((snd\ (clist!i))!j) = snd((snd\ (clist!i'))!j) \wedge$
$snd((snd\ (clist!i))!Suc\ j) = snd((snd\ (clist!i'))!Suc\ j)$
by *fastforce*
then have *clist-i-Guardxs*:
$(snd((snd\ (clist!i))!j),\ snd((snd\ (clist!i))!Suc\ j)) \in$
$Guar(xs!i')$
using *clist-i′-Guardxs* **by** *fastforce*
thus *?thesis*
using *False a20 a4* **by** *fastforce*
qed
next
assume *a10*:$(\Gamma\vdash_p(l!j)\ \rightarrow_e\ (l!(Suc\ j)) \wedge$
$(\forall\, i{<}length\ clist.\ \ \Gamma\vdash_c\ (snd\ (clist!i))!j\ \rightarrow_e\ ((snd\ (clist!i))!(Suc\ j))))$
then have $(snd\ (l\ !\ j),\ snd\ (l\ !\ Suc\ j)) \in R$
using *a00 a10 a6* **unfolding** *par-assum-def* **by** *fastforce*
then show *?thesis* **using** *a8 a00*
unfolding *conjoin-def same-state-def*
by *fastforce*
qed
} **thus** *?thesis* **by** *blast*
qed

definition *tran-True* **where** *tran-True* $\equiv$ *True*

definition *after* **where** *after* $\equiv$ *True*

lemma *four*:
$[\![xs{\neq}[];\ \ \forall\, i{<}length\ xs.\ \ R \cup (\bigcup j{\in}\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq\ (Rely\ (xs\ !\ i));$
$(\bigcup j{<}length\ xs.\ \ (Guar\ (xs\ !\ j))) \subseteq (G);$
$p \subseteq (\bigcap i{<}length\ xs.\ \ (Pre\ (xs\ !\ i)));$
$\forall\, i{<}length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),\ \ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)];$
$(\Gamma,l) \in par\text{-}cp\ \Gamma\ (ParallelCom\ xs)\ s;\ (\Gamma,l) \in par\text{-}assum(p,\ R);\ Suc\ i < length\ l;$
$\Gamma\vdash_p\ (l!i) \rightarrow (l!(Suc\ i));$
$(\forall\, (c,p,R,G,q,a){\in}\ \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a]);$
$snd\ (last\ l) \notin Fault\ `\ F]\!]$
$\Longrightarrow (snd\ (l\ !\ i),\ snd\ (l\ !\ Suc\ i)) \in G$
proof −
assume *a0*:$xs{\neq}[]$ **and**
a1:$\forall\, i{<}length\ xs.\ \ R \cup (\bigcup j{\in}\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq\ (Rely\ (xs\ !\ i))$ **and**
a2:$(\bigcup j{<}length\ xs.\ \ (Guar\ (xs\ !\ j))) \subseteq (G)$ **and**
a3:$p \subseteq (\bigcap i{<}length\ xs.\ \ (Pre\ (xs\ !\ i)))$ **and**
a4:$\forall\, i{<}length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),\ \ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)]$ **and**

$a5$:$(\Gamma,l) \in$ *par-cp* Γ (*ParallelCom xs*) *s* **and**
$a6$:$(\Gamma,l) \in$ *par-assum*$(p,\ R)$ **and**
$a7$: *Suc i* $<$ *length l* **and**
$a8$:$\Gamma\vdash_p (l!i) \rightarrow (l!(Suc\ i))$ **and**
$a10$:$(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a])$ **and**
$a11$:*snd* (*last l*) $\notin$ *Fault ‘ F*
have *length-par-xs*:*length* (*ParallelCom xs*) = *length xs* **unfolding** *ParallelCom-def* **by** *fastforce*
then have (*ParallelCom xs*)$\neq$[] **using** *a0* **by** *fastforce*
then have $(\Gamma,l) \in\{(\Gamma 1,c).\ \exists$ *clist*. (*length clist*)=(*length* (*ParallelCom xs*)) $\wedge$
($\forall$ *i*<*length clist*. (*clist*!*i*) $\in$ *cp* Γ ((*ParallelCom xs*)!*i*) *s*) $\wedge$ $(\Gamma,c) \propto$ *clist* $\wedge$ $\Gamma 1=\Gamma\}$
using *one a5* **by** *fastforce*
then obtain *clist* **where** (*length clist*)=(*length xs*) $\wedge$
($\forall$ *i*<*length clist*. (*clist*!*i*) $\in$ *cp* Γ ((*ParallelCom xs*)!*i*) *s*) $\wedge$ $(\Gamma,l) \propto$ *clist*
using *length-par-xs* **by** *auto*
then have *conjoin*:(*length clist*)=(*length xs*) $\wedge$
($\forall$ *i*<*length clist*. (*clist*!*i*) $\in$ *cp* Γ (*Com* (*xs* ! *i*)) *s*) $\wedge$ $(\Gamma,l) \propto$ *clist*
using *ParallelCom-Com* **by** *fastforce*
then have *length-xs-clist*:*length xs* = *length clist* **by** *auto*
have *clist-cp*:$\forall$ *i*<*length clist*. (*clist*!*i*) $\in$ *cp* Γ (*Com* (*xs* ! *i*)) *s* **using** *conjoin* **by** *auto*
have *conjoin*:$(\Gamma,l) \propto$ *clist* **using** *conjoin* **by** *auto*
have *l-not-empty*:*l*$\neq$[] **using** *a5 par-cptn.simps* **unfolding** *par-cp-def* **by** *fastforce*
then have *l-g0*:*0*<*length l* **by** *fastforce*
then have *last-l*:*last l* = *l*!((*length l*) $-$ *1*) **by** (*simp add*: *last-conv-nth*)
have $\forall$ *i*< *length l*. *fst* (*l*!*i*) = *map* (λx. *fst* ((*snd x*)!*i*)) *clist*
using *conjoin* **unfolding** *conjoin-def same-program-def* **by** *fastforce*
obtain *Ps si Ps$'$ ssi* **where** *li*:*l*!*i* = (*Ps*,*si*) $\wedge$ *l*!(*Suc i*) = (*Ps$'$*, *ssi*) **by** *fastforce*
then have $\exists$ *j r*. *j*<*length Ps* $\wedge$ *Ps$'$* = *Ps*[*j*:=*r*] $\wedge$ $(\Gamma\vdash_c((Ps!j),si) \rightarrow (r,\ ssi))$
using *a8 par-ctranE* **by** *fastforce*
then obtain *j r* **where** *step-c*:*j*<*length Ps* $\wedge$ *Ps$'$* = *Ps*[*j*:=*r*] $\wedge$ $(\Gamma\vdash_c((Ps!j),si) \rightarrow (r,\ ssi))$
by *auto*
have *length-Ps-clist*:
length Ps = *length clist* $\wedge$ *length Ps* = *length Ps$'$*
using *conjoin a7 conjoin-same-length li step-c* **by** *fastforce*
have *from-step*:(*snd* (*clist*!*j*))!*i* = ((*Ps*!*j*),*si*) $\wedge$ (*snd* (*clist*!*j*))!(*Suc i*) = (*Ps$'$*!*j*,*ssi*)

proof $-$
have *f2*: *Ps* = *fst* (*snd* (Γ, *l*) ! *i*) **and** *f2$'$*:*Ps$'$* = *fst* (*snd* (Γ, *l*) ! (*Suc i*))
using *li* **by** *auto*
have *f3*:*si* = *snd* (*snd* (Γ, *l*) ! *i*) $\wedge$ *ssi* = *snd* (*snd* (Γ, *l*) ! (*Suc i*))
by (*simp add*: *li*)
then have (*snd* (*clist*!*j*))!*i* = ((*Ps*!*j*),*si*)
using *f2 conjoin a7 step-c* **unfolding** *conjoin-def same-program-def same-state-def* **by** *force*
moreover have (*snd* (*clist*!*j*))!(*Suc i*) = (*Ps$'$*!*j*,*ssi*)
using *f2$'$ f3 conjoin a7 step-c length-Ps-clist*

unfolding *conjoin-def same-program-def same-state-def*
by *auto*
ultimately show *?thesis* **by** *auto*
qed
then have *step-clist*:$\Gamma\vdash_c(snd\ (clist!j))!i \rightarrow (snd\ (clist!j))!(Suc\ i)$
using *from-step step-c* **by** *fastforce*
have *j-xs*:$j<length\ xs$ **using** *step-c length-Ps-clist length-xs-clist* **by** *auto*
have $j<length\ clist$ **using** *j-xs length-xs-clist* **by** *auto*
also have
$\forall i\ j\ ns\ ns'.\ j < length\ clist \wedge Suc\ i < length\ l \longrightarrow$
$\Gamma\vdash_c snd\ (clist\ !\ j)\ !\ i \rightarrow snd\ (clist\ !\ j)\ !\ Suc\ i \longrightarrow$
$(snd\ (snd\ (clist\ !\ j)\ !\ i),\ snd\ (snd\ (clist\ !\ j)\ !\ Suc\ i)) \in Guar\ (xs\ !\ j)$
using *two*[*OF a1 a3 a4 length-xs-clist a5 a6 clist-cp conjoin a10 a11*] **by** *auto*
ultimately have $(snd\ (snd\ (clist\ !\ j)\ !\ i),\ snd\ (snd\ (clist\ !\ j)\ !\ Suc\ i)) \in Guar\ (xs\ !\ j)$
using *a7 step-c length-Ps-clist step-clist* **by** *metis*
then have $(snd\ (l!i),\ snd\ (l!(Suc\ i)))\in Guar\ (xs\ !\ j)$
using *from-step a2 length-xs-clist step-c li* **by** *fastforce*
then show *?thesis* **using** *a2 j-xs*
unfolding *sep-conj-def tran-True-def after-def Satis-def* **by** *fastforce*
qed

lemma *same-program-last*:$l\neq[] \Longrightarrow (\Gamma,l) \propto clist \Longrightarrow i<length\ clist \Longrightarrow fst\ (last\ (snd\ (clist!i))) = fst\ (last\ l)\ !\ i$
proof −
assume *l-not-empty*:$l\neq[]$ **and**
conjoin: $(\Gamma,l) \propto clist$ **and**
i-clist: $i<length\ clist$
have *last-clist-eq-l*:$\forall i<length\ clist.\ last\ (snd\ (clist!i)) = (snd\ (clist!i))!((length\ l) - 1)$
using *conjoin last-conv-nth l-not-empty*
unfolding *conjoin-def same-length-def*
by (*metis length-0-conv snd-eqD*)
then have *last-l*:$last\ l = l!((length\ l)-1)$ **using** *l-not-empty* **by** (*simp add*: *last-conv-nth*)
have $fst\ (last\ l) = map\ (\lambda x.\ fst\ (snd\ x\ !\ ((length\ l)-1)))\ clist$
using *l-not-empty last-l conjoin* **unfolding** *conjoin-def same-program-def* **by** *auto*
also have $(map\ (\lambda x.\ fst\ (snd\ x\ !\ ((length\ l)-1)))\ clist)!i =$
$fst\ ((snd\ (clist!i))!\ ((length\ l)-1))$ **using** *i-clist* **by** *fastforce*
also have $fst\ ((snd\ (clist!i))!\ ((length\ l)-1)) =$
$fst\ ((snd\ (clist!i))!\ ((length\ (snd\ (clist!i)))-1))$
using *conjoin i-clist* **unfolding** *conjoin-def same-length-def* **by** *fastforce*
also then have $fst\ ((snd\ (clist!i))!\ ((length\ (snd\ (clist!i)))-1)) = fst\ (last\ (snd\ (clist!i)))$
using *i-clist l-not-empty conjoin last-clist-eq-l last-conv-nth* **unfolding** *conjoin-def same-length-def*
by *presburger*
finally show *?thesis* **by** *auto*

qed

lemma *five*:
$[\![xs\neq[]; \ \forall i<length\ xs.\ \ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i));$
$p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)));$
$(\bigcap i<length\ xs.\ (Post\ (xs\ !\ i))) \subseteq q;$
$(\bigcup i<length\ xs.\ (Abr\ (xs\ !\ i))) \subseteq a\ ;$
$\forall i < length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)];$
$(\Gamma,l) \in par\text{-}cp\ \Gamma\ (ParallelCom\ xs)\ s;\ (\Gamma,l) \in par\text{-}assum(p,\ R);$
$All\text{-}End\ (last\ l);\ snd\ (last\ l) \notin Fault\ `\ F;(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a])\ ]\!] \Longrightarrow$
$(\exists j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Throw \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)) \vee$
$(\forall j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Skip \wedge$
$snd\ (last\ l) \in Normal\ `\ q)$

proof−
assume *a0*:$xs\neq[]$ **and**
a1:$\forall i<length\ xs.\ \ R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq\ (Rely\ (xs\ !\ i))$ **and**
a2:$p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i)))$ **and**
a3:$(\bigcap i<length\ xs.\ (Post\ (xs\ !\ i))) \subseteq q$ **and**
a4:$(\bigcup i<length\ xs.\ (Abr\ (xs\ !\ i))) \subseteq a$ **and**
a5:$\forall i < length\ xs.$
$\Gamma,\Theta \models_{/F}\ Com\ (xs\ !\ i)\ sat\ [Pre\ (xs!i),$
$Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),$
$Post\ (xs\ !\ i),Abr\ (xs\ !\ i)]$ **and**
a6:$(\Gamma,l) \in par\text{-}cp\ \Gamma\ (ParallelCom\ xs)\ s$ **and**
a7:$(\Gamma,l) \in par\text{-}assum(p,\ R)$**and**
a8:$All\text{-}End\ (last\ l)$ **and**
a9:$snd\ (last\ l) \notin Fault\ `\ F$ **and**
a10:$(\forall (c,p,R,G,q,a)\in \Theta.\ \Gamma \models_{/F} (Call\ c)\ sat\ [p,\ R,\ G,\ q,a])$
have *length-par-xs*:$length\ (ParallelCom\ xs) = length\ xs$ **unfolding** *ParallelCom-def* **by** *fastforce*
then have $(ParallelCom\ xs)\neq[]$ **using** *a0* **by** *fastforce*
then have $(\Gamma,l) \in\{(\Gamma 1,c).\ \exists clist.\ (length\ clist)=(length\ (ParallelCom\ xs)) \wedge$
$(\forall i<length\ clist.\ (clist!i) \in cp\ \Gamma\ ((ParallelCom\ xs)!i)\ s) \wedge (\Gamma,c) \propto clist \wedge \Gamma 1=\Gamma\}$
using *one a6* **by** *fastforce*
then obtain *clist* **where** $(length\ clist)=(length\ xs) \wedge$
$(\forall i<length\ clist.\ (clist!i) \in cp\ \Gamma\ ((ParallelCom\ xs)!i)\ s) \wedge (\Gamma,l) \propto clist$
using *length-par-xs* **by** *auto*
then have *conjoin*:$(length\ clist)=(length\ xs) \wedge$
$(\forall i<length\ clist.\ (clist!i) \in cp\ \Gamma\ \ (Com\ (xs\ !\ i))\ s) \wedge (\Gamma,l) \propto clist$
using *ParallelCom-Com* **by** *fastforce*

then have *length-xs-clist*:*length xs* = *length clist* **by** *auto*

have *clist-cp*:∀ *i*<*length clist*. (*clist*!*i*) ∈ *cp* Γ (*Com* (*xs* ! *i*)) *s* **using** *conjoin* **by** *auto*
have *conjoin*:(Γ,*l*) ∝ *clist* **using** *conjoin* **by** *auto*
have *l-not-empty*:*l*≠[] **using** *a6 par-cptn.simps* **unfolding** *par-cp-def* **by** *fastforce*
then have *l-g0*:*0*<*length l* **by** *fastforce*
then have *last-l*:*last l* = *l*!((*length l*) − *1*) **by** (*simp add*: *last-conv-nth*)
have *clist-assum*:∀ *i*<*length clist*. (*clist*!*i*) ∈ *assum* (*Pre* (*xs*!*i*),*Rely* (*xs*!*i*))
proof −
{ **fix** *i*
assume *i-length*:*i*<*length clist*
obtain Γ*1 li* **where** *clist*:*clist*!*i*=(Γ*1*,*li*) **by** *fastforce*
then have Γ*eq*:Γ*1*=Γ
using *conjoin i-length* **unfolding** *conjoin-def same-functions-def* **by** *fastforce*
have (Γ*1*,*li*) ∈ *assum* (*Pre* (*xs*!*i*),*Rely* (*xs*!*i*))
proof−
have *l*:*snd* (*li*!*0*) ∈ *Normal* ‘ ((*Pre* (*xs*!*i*)))
proof −
have *snd-l*:*snd* (Γ,*l*) = *l* **by** *fastforce*
have *snd* (*l*!*0*) ∈ *Normal* ‘ (*p*)
using *a7* **unfolding** *par-assum-def* **by** *fastforce*
also have *snd* (*l*!*0*) = *snd* (*li*!*0*)
using *i-length conjoin l-g0 clist*
unfolding *conjoin-def same-state-def* **by** *fastforce*
finally show *?thesis* **using** *a2 i-length length-xs-clist*
by *auto*
qed
have *r*:(∀ *j*. *Suc j* < *length li* ⟶
$\Gamma\vdash_c(li!j) \rightarrow_e (li!(Suc\ j)) \longrightarrow$
(*snd*(*li*!*j*), *snd*(*li*!(*Suc j*))) ∈ *Rely* (*xs*!*i*))
using *three*[*OF a0 a1 a2 a5 length-xs-clist a6 a7 clist-cp conjoin a10 a9*]
i-length conjoin a1 length-xs-clist clist
unfolding *assum-def conjoin-def same-length-def* **by** *fastforce*
show *?thesis* **using** *l r* Γ*eq* **unfolding** *assum-def* **by** *fastforce*
qed
then have *clist*!*i* ∈ *assum* (*Pre* (*xs*!*i*),*Rely* (*xs*!*i*)) **using** *clist* **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
then have *clist-com*:∀ *i*<*length clist*. (*clist*!*i*) ∈ *comm* (*Guar* (*xs*!*i*),(*Post*(*xs*!*i*),*Abr* (*xs*!*i*))) *F*
using *a5* **unfolding** *com-cvalidity-def*
using *a10* **unfolding** *com-validity-def* **using** *clist-cp length-xs-clist*
by *force*
have *last-clist-eq-l*:∀ *i*<*length clist*. *last* (*snd* (*clist*!*i*)) = (*snd* (*clist*!*i*))!((*length l*) − *1*)
using *conjoin last-conv-nth l-not-empty*
unfolding *conjoin-def same-length-def*
by (*metis length-0-conv snd-eqD*)

then have *last-clist-l*:∀ *i*<*length clist*. *snd* (*last* (*snd* (*clist*!*i*))) = *snd* (*last l*)
using *last-l conjoin l-not-empty* **unfolding** *conjoin-def same-state-def same-length-def*

by *simp*
show *?thesis*
proof(*cases* ∀ *i*<*length* (*fst* (*last l*)). *fst* (*last l*)!*i* = *Skip*)
assume *ac1*:∀ *i*<*length* (*fst* (*last l*)). *fst* (*last l*)!*i* = *Skip*
have (∀ *j*<*length* (*fst* (*last l*)). *fst* (*last l*) ! *j* = *LanguageCon.com.Skip* ∧ *snd* (*last l*) ∈ *Normal* ' *q*)
proof −
{**fix** *j*
assume *aj*:*j*<*length* (*fst* (*last l*))
have ∀ *i*<*length clist*. *snd* (*last* (*snd* (*clist*!*i*))) ∈ *Normal* ' *Post*(*xs*!*i*)
proof−
{**fix** *i*
assume *a20*:*i*<*length clist*
then have *snd-last*:*snd* (*last* (*snd* (*clist*!*i*))) = *snd* (*last l*)
using *last-clist-l* **by** *fastforce*
have *last-clist-not-F*:*snd* (*last* (*snd* (*clist*!*i*)))∉ *Fault* ' *F*
using *a9 last-clist-l a20* **by** *fastforce*
have *fst* (*last l*) ! *i* = *Skip*
using *a20 ac1 conjoin-same-length*[*OF conjoin*]
by (*simp add*: *l-not-empty last-l*)
also have *fst* (*last l*) ! *i*=*fst* (*last* (*snd* (*clist*!*i*)))
using *same-program-last*[*OF l-not-empty conjoin a20*] **by** *auto*
finally have *fst* (*last* (*snd* (*clist*!*i*))) = *Skip* **.**
then have *snd* (*last* (*snd* (*clist*!*i*))) ∈ *Normal* ' *Post*(*xs*!*i*)
using *clist-com last-clist-not-F a20*
unfolding *comm-def final-def* **by** *fastforce*
} **thus** *?thesis* **by** *auto*
qed
then have ∀ *i*<*length xs*. *snd* (*last l*) ∈ *Normal* ' *Post*(*xs*!*i*)
using *last-clist-l length-xs-clist* **by** *fastforce*
then have ∀ *i*<*length xs*. ∃ *x*∈(*Post*(*xs*!*i*)). *snd* (*last l*) = *Normal x*
by *fastforce*
moreover have ∀ *t*. (∀ *i*<*length xs*. *t*∈ *Post* (*xs* ! *i*))⟶ *t*∈ *q* **using** *a3*
by *fastforce*
ultimately have (∃ *x*∈ *q*. *snd* (*last l*) = *Normal x*) **using** *a0*
by (*metis* (*mono-tags*, *lifting*) *length-greater-0-conv xstate.inject*(*1*))
then have *snd* (*last l*) ∈ *Normal* ' *q* **by** *fastforce*
then have *fst* (*last l*) ! *j* = *LanguageCon.com.Skip* ∧ *snd* (*last l*) ∈ *Normal* ' *q*
using *aj ac1* **by** *fastforce*
} **thus** *?thesis* **by** *auto*
qed
thus *?thesis* **by** *auto*
next
assume ¬ (∀ *i*<*length* (*fst* (*last l*)). *fst* (*last l*)!*i* = *Skip*)
then obtain *i* **where** *a20*:*i*< *length* (*fst* (*last l*)) ∧ *fst* (*last l*)!*i* ≠ *Skip*

by *fastforce*
then have *last-i-throw*:*fst* (*last* *l*)!*i* = *Throw* ∧ (∃ *n*. *snd* (*last* *l*) = *Normal* *n*)
using *a8* **unfolding** *All-End-def final-def* **by** *fastforce*
have *length* (*fst* (*last* *l*)) = *length clist*
using *conjoin-same-length*[*OF conjoin*] *l-not-empty last-l*
by *simp*
then have *i-length*:*i*<*length clist* **using** *a20* **by** *fastforce*
then have *snd-last*:*snd* (*last* (*snd* (*clist*!*i*))) = *snd* (*last* *l*)
using *last-clist-l* **by** *fastforce*
have *last-clist-not-F*:*snd* (*last* (*snd* (*clist*!*i*)))∉ *Fault* ‘ *F*
using *a9 last-clist-l i-length* **by** *fastforce*
then have *fst* (*last* (*snd* (*clist*!*i*))) = *fst* (*last* *l*) ! *i*
using *i-length same-program-last* [*OF l-not-empty conjoin*] **by** *fastforce*
then have *fst* (*last* (*snd* (*clist*!*i*))) = *Throw*
using *last-i-throw* **by** *fastforce*
then have *snd* (*last* (*snd* (*clist*!*i*))) ∈ *Normal* ‘ *Abr*(*xs*!*i*)
using *clist-com last-clist-not-F i-length last-i-throw snd-last*
unfolding *comm-def final-def* **by** *fastforce*
then have *snd* (*last* *l*)∈ *Normal* ‘ *Abr*(*xs*!*i*) **using** *last-clist-l i-length*
by *fastforce*
then have *snd* (*last* *l*)∈ *Normal* ‘ (*a*) **using** *a4 a0 i-length length-xs-clist* **by** *fastforce*
then have ∃ *j*<*length* (*fst* (*last* *l*)).
fst (*last* *l*) ! *j* = *LanguageCon.com.Throw* ∧ *snd* (*last* *l*) ∈ *Normal* ‘ *a*
using *last-i-throw a20* **by** *fastforce*
thus *?thesis* **by** *auto*
qed
qed

lemma *ParallelEmpty* [*rule-format*]:
∀ *i s*. (Γ,*l*) ∈ *par-cp* Γ (*ParallelCom* []) *s* ⟶
Suc i < *length l* ⟶ ¬ (Γ ⊢$_p$ (*l*!*i*) → (*l*!*Suc i*))
apply(*induct-tac l*)
apply *simp*
apply *clarify*
apply(*case-tac list*,*simp*,*simp*)
apply(*case-tac i*)
apply(*simp add*:*par-cp-def ParallelCom-def*)
apply(*erule par-ctranE*,*simp*)
apply(*simp add*:*par-cp-def ParallelCom-def*)
apply *clarify*
apply(*erule par-cptn.cases*,*simp*)
apply *simp*
by (*metis list.inject list.size*(*3*) *not-less0 step-p-pair-elim-cases*)

lemma *ParallelEmpty2*:
assumes *a0*:(Γ,*l*) ∈ *par-cp* Γ (*ParallelCom* []) *s* **and**

a1: $i < length\ l$
shows $fst\ (l!i) = []$
proof –
have $paremp$:$ParallelCom\ [] = []$ **unfolding** *ParallelCom-def* **by** *auto*
then have $l0$:$l!0 = ([],s)$ **using** *a0* **unfolding** *par-cp-def* **by** *auto*
then have $(\Gamma,l) \in par\text{-}cptn$ **using** *a0* **unfolding** *par-cp-def* **by** *fastforce*
thus *?thesis* **using** *l0 a1*
proof (*induct arbitrary*: *i s*)
case *ParCptnOne* **thus** *?case* **by** *auto*
next
case $(ParCptnEnv\ \Gamma\ P\ s1\ t\ xs\ i\ s)$
thus *?case*
proof –
have $f1$: $i < Suc\ (Suc\ (length\ xs))$
using *ParCptnEnv.prems*(*2*) **by** *auto*
have $(P,\ s1) = ([],\ s)$
using *ParCptnEnv.prems*(*1*) **by** *auto*
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *ParCptnEnv.hyps*(*3*) *diff-Suc-1 fst-conv length-Cons less-Suc-eq-0-disj nth-Cons'*)
qed
next
case $(ParCptnComp\ \Gamma\ P\ s1\ Q\ t\ xs)$
have $(\Gamma,\ (P,s1)\#(Q,\ t)\ \#\ xs) \in par\text{-}cp\ \Gamma\ (ParallelCom\ [])\ s1$
using *ParCptnComp*(*4*) *ParCptnComp*(*1*) *step-p-elim-cases* **by** *fastforce*
then have $\neg\ \Gamma\vdash_p\ (P,\ s1) \rightarrow (Q,\ t)$ **using** *ParallelEmpty ParCptnComp* **by** *fastforce*
thus *?case* **using** *ParCptnComp* **by** *auto*
qed
qed

lemma *parallel-sound*:
$\forall i<length\ xs.$
$R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq (Rely\ (xs\ !\ i)) \Longrightarrow$
$(\bigcup j<length\ xs.\ (Guar\ (xs\ !\ j))) \subseteq G \Longrightarrow$
$p \subseteq (\bigcap i<length\ xs.\ (Pre\ (xs\ !\ i))) \Longrightarrow$
$(\bigcap i<length\ xs.\ (Post\ (xs\ !\ i))) \subseteq q \Longrightarrow$
$(\bigcup i<length\ xs.\ (Abr\ (xs\ !\ i))) \subseteq a \Longrightarrow$
$\forall i<length\ xs.$
$\Gamma,\Theta \models_{/F} Com\ (xs\ !i)\ sat\ [Pre\ (xs\ !i),\ Rely\ (xs\ !\ i),\ Guar\ (xs\ !\ i),\ Post\ (xs\ !\ i),Abr\ (xs\ !\ i)] \Longrightarrow$
$\Gamma,\Theta \models_{/F} ParallelCom\ xs\ SAT\ [p,\ R,\ G,\ q,a]$

proof –
assume
$a0$:$\forall i<length\ xs.$
$R \cup (\bigcup j\in\{j.\ j < length\ xs \wedge j \neq i\}.\ (Guar\ (xs\ !\ j)))$
$\subseteq\ (Rely\ (xs\ !\ i))$ **and**

a1:($\bigcup$*j*<*length xs*. (*Guar* (*xs* ! *j*))) ⊆ *G* **and**
a2:*p* ⊆ ($\bigcap$*i*<*length xs*. (*Pre* (*xs* ! *i*))) **and**
a3:($\bigcap$*i*<*length xs*. (*Post* (*xs* ! *i*))) ⊆ *q* **and**
a4:($\bigcup$*i*<*length xs*. (*Abr* (*xs* ! *i*))) ⊆ *a* **and**
a5:∀ *i*<*length xs*.
Γ,Θ $\models_{/F}$ *Com* (*xs* !*i*) *sat* [*Pre* (*xs* !*i*), *Rely* (*xs* ! *i*), *Guar* (*xs* ! *i*), *Post* (*xs* ! *i*),*Abr* (*xs* ! *i*)]
{
assume *a00*:(∀ (*c*,*p*,*R*,*G*,*q*,*a*)∈ Θ. Γ $\models_{/F}$ (*Call c*) *sat* [*p*, *R*, *G*, *q*,*a*])
{ fix *s l*
assume *a10*: (Γ,*l*) ∈ *par-cp* Γ (*ParallelCom xs*) *s* ∧ (Γ,*l*) ∈ *par-assum*(*p*, *R*)
then have *c-par-cp*:(Γ,*l*) ∈ *par-cp* Γ (*ParallelCom xs*) *s* **by** *auto*
have *c-par-assum*: (Γ,*l*) ∈ *par-assum*(*p*, *R*) **using** *a10* **by** *auto*
{ fix *i ns ns′*
assume *a20*:*snd* (*last l*) ∉ *Fault* ‘ *F*
{
assume *a30*:*Suc i*<*length l* **and**
a31: Γ$\vdash_p$(*l*!*i*) → (*l*!(*Suc i*))
have *xs-not-empty*:*xs*≠[]
proof −
{
assume *xs* = []
then have ¬ (Γ $\vdash_p$ (*l*!*i*) → (*l*!*Suc i*))
using *a30 a10 ParallelEmpty* **by** *fastforce*
then have *False* **using** *a31* **by** *auto*
} thus *?thesis* **by** *auto*
qed
then have (*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G*
using *four*[*OF xs-not-empty a0 a1 a2 a5 c-par-cp c-par-assum a30 a31 a00 a20*] **by** *blast*

} then have *Suc i*<*length l* ⟶
Γ$\vdash_p$(*l*!*i*) → (*l*!(*Suc i*)) ⟶
(*snd*(*l*!*i*), *snd*(*l*!(*Suc i*))) ∈ *G* **by** *auto*
note *l* = *this*
{ assume *a30*:*All-End* (*last l*)
then have *xs-not-empty*:*xs*≠[]
proof −
{ assume *xs-emp*:*xs*=[]
have *lenl*:*0*<*length l* **using** *a10* **unfolding** *par-cp-def* **using** *par-cptn.simps* **by** *fastforce*
then have (*length l*) − *1* < *length l* **by** *fastforce*
then have *fst*(*l*!((*length l*) − *1*)) = [] **using** *ParallelEmpty2 a10 xs-emp* **by** *fastforce*
then have *False* **using** *a30 lenl* **unfolding** *All-End-def*
by (*simp add*: *last-conv-nth*)
} thus *?thesis* **by** *auto*
qed

then have $(\exists j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Throw\ \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)) \vee$
$(\forall j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Skip\ \wedge$
$snd\ (last\ l) \in Normal\ `\ q)$
using *five*[*OF xs-not-empty a0 a2 a3 a4 a5 c-par-cp c-par-assum a30 a20 a00*] **by** *blast*
} **then have** *All-End* (*last l*) $\longrightarrow$
$(\exists j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Throw\ \wedge$
$snd\ (last\ l) \in Normal\ `\ (a)) \vee$
$(\forall j<length\ (fst\ (last\ l)).\ fst\ (last\ l)!j=Skip\ \wedge$
$snd\ (last\ l) \in Normal\ `\ q)$ **by** *auto*
note *res1* = *conjI*[*OF l this*]
}
then have $(\Gamma,l) \in par\text{-}comm(G,\ (q,a))\ F$ **unfolding** *par-comm-def* **by** *auto*

}
then have $\Gamma \models_{/F}$ (*ParallelCom xs*) *SAT* $[p,\ R,\ G,\ q,a]$
unfolding *par-com-validity-def par-cp-def* **by** *fastforce*
} **thus** *?thesis* **using** *par-com-cvalidity-def* **by** *fastforce*
qed

theorem
par-rgsound:$\Gamma,\Theta \vdash_{/F}$ *Ps SAT* $[p,\ R,\ G,\ q,a] \Longrightarrow$
$\Gamma,\Theta \models_{/F}$ (*ParallelCom Ps*) *SAT* $[p,\ R,\ G,\ q,a]$
proof (*induction rule*:*par-rghoare.induct*)
case (*Parallel xs R G p q a* $\Gamma\ \Theta$ *F*)
thus *?case* **using** *localRG-sound com-cnvalid-to-cvalid parallel-sound*[*of xs R G p q a* $\Gamma\ \Theta$ *F*]
by *fast*
qed
lemma *Conseq'*:$\forall s.\ s\in p \longrightarrow$
$(\exists p'\ q'\ a'\ R'\ G'.$
$(\forall Z.\ \Gamma,\Theta\vdash_{/F} P\ sat\ [(p'\ Z),\ (R'\ Z),\ (G'\ Z),\ (q'\ Z),(a'\ Z)]) \wedge$
$(\exists\ Z.\ s\in p'\ Z \wedge (q'\ Z \subseteq q) \wedge (a'\ Z \subseteq a) \wedge (G'\ Z \subseteq G) \wedge (R \subseteq R'\ Z)))$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ sat\ [p,\ R,\ G,\ q,a]$
apply (*rule Conseq*)
by (*meson order-refl*)

lemma *conseq*:$[\![\forall Z.\ \Gamma,\Theta'\vdash_{/F} P\ sat\ [(p'\ Z),\ (R'\ Z),\ (G'\ Z),\ (q'\ Z),(a'\ Z)];\ \Theta' \subseteq \Theta\ ;$
$\forall s.\ s \in p \longrightarrow (\exists\ Z.\ s\in p'\ Z \wedge (q'\ Z \subseteq q) \wedge (a'\ Z \subseteq a) \wedge (G'\ Z \subseteq G) \wedge (R \subseteq R'\ Z))]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ sat\ [p,\ R,\ G,\ q,a]$
by (*rule Conseq*) (*meson order-refl*)

lemma *conseqPrePost*[*trans*]:
$\Gamma,\Theta' \vdash_{/F}$ *P sat* $[p', R', G', q', a'] \Longrightarrow \Theta' \subseteq \Theta \Longrightarrow$
$p \subseteq p' \Longrightarrow q' \subseteq q \Longrightarrow a' \subseteq a \Longrightarrow G' \subseteq G \Longrightarrow R \subseteq R' \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, q, a]$
by (*rule conseq*) *auto*

lemma *conseqPre*[*trans*]:
$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p', R, G, q, a] \Longrightarrow$
$p \subseteq p' \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, q, a]$
by (*rule conseq*) *auto*

lemma *conseqPost*[*trans*]:
$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, q', a'] \Longrightarrow$
$q' \subseteq q \Longrightarrow a' \subseteq a \Longrightarrow$
$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, q, a]$
by (*rule conseq*) *auto*

lemma shows *x*:$\exists$ (*sa'*::*nat set*). ($\forall$ *x*. (*x*$\in$ *sa*) = ((*to-nat x*) $\in$ *sa'*))
by (*metis* (*mono-tags*, *hide-lams*) *from-nat-to-nat imageE image-eqI*)

lemma *not-empty-set-countable*:
assumes *a0*:*sa*$\neq$({}::('*a*::*countable*) *set*)
shows {*i*. ((λ*i*. *i*$\in$ *sa*) *o from-nat*) *i*}$\neq${}
by (*metis* (*full-types*) *Collect-empty-eq-bot assms comp-apply empty-def equals0I from-nat-to-nat*)

lemma *eq-set-countable*:($\bigcap$ *i*$\in${*i*. ((λ*i*. *i*$\in$ *sa*) *o from-nat*) *i*}. (*q o from-nat*) *i*) = (($\bigcap$ *i*$\in$*sa*. *q i*))
apply *auto*
by (*metis* (*no-types*) *from-nat-to-nat*)

lemma *conj-inter-countable*[*trans*]:
assumes *a0*:*sa*$\neq$({}::('*a*::*countable*) *set*) **and**
a1:$\forall$ *i*$\in$*sa*. $\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, q\ i, a]$
shows$\Gamma,\Theta \vdash_{/F}$ *P sat* $[p, R, G, (\bigcap i \in sa.\ q\ i), a]$
proof−
have $\forall$ *i*$\in${*i*. ((λ*i*. *i*$\in$ *sa*) *o from-nat*) *i*}. $\Gamma,\Theta \vdash_{/F}$ *P sat* [*p*, *R*, *G*,(*q o from-nat*) *i*,*a*]
using *a1* **by** *auto*
then have $\Gamma,\Theta \vdash_{/F}$ *P sat* [*p*, *R*, *G*,$\bigcap$ *i*$\in${*i*. ((λ*i*. *i*$\in$ *sa*) *o from-nat*) *i*}. (*q o from-nat*) *i*,*a*]
using *Conj-Inter*[*OF not-empty-set-countable*[*OF a0*]] **by** *auto*
thus *?thesis* **using** *eq-set-countable*
by *metis*
qed

lemma *all-Post*[*trans*]:
  assumes *a0*:∀ *p-n*::(′*a*::*countable*). Γ,Θ⊢$_{/F}$ *C sat* [*P*, *R*, *G*, *Q p-n*, *Qa*]
  showsΓ,Θ⊢$_{/F}$ *C sat* [*P*, *R*, *G*,{*s*. ∀ *p-n*. *s*∈*Q p-n*},*Qa*]
proof−
  have Γ,Θ⊢$_{/F}$ *C sat* [*P*, *R*, *G*,(⋂ *p-n*. *Q p-n*),*Qa*]
    using *a0 conj-inter-countable*[*of UNIV*] **by** *auto*
  moreover have *s1*:∀ *P*. {*s*. ∀ *p-n*. *s*∈*P p-n*} = (⋂ *p-n*. *P p-n*)
   by *auto*
  ultimately show *?thesis*
   by (*simp add*: *s1*)
qed

lemma *all-Pre*[*trans*]:
  assumes *a0*:∀ *p-n*. Γ,Θ⊢$_{/F}$ *C sat* [*P p-n*, *R*, *G*, *Q*, *Qa*]
  showsΓ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,*Q*,*Qa*]
proof−
  {**fix** *p-n*
   have Γ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,*Q*,*Qa*]
   proof−
     have {*v*. ∀ *n*. *v* ∈ *P n*} ⊆ *P p-n* **by** *force*
    then show *?thesis* **by** (*meson a0 LocalRG-HoareDef.conseqPrePost subset-eq*)
   qed
  } **thus** *?thesis* **by** *auto*
qed

lemma *Pre-Post-all*:
  assumes *a0*:∀ *p-n*::(′*a*::*countable*). Γ,Θ⊢$_{/F}$ *C sat* [*P p-n*, *R*, *G*, *Q p-n*, *Qa*]
  showsΓ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,{*s*. ∀ *p-n*. *s*∈*Q p-n*},*Qa*]
proof−
  {**fix** *p-n*

   have Γ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,*Q p-n*,*Qa*]
   proof−
     have {*v*. ∀ *n*. *v* ∈ *P n*} ⊆ *P p-n* **by** *force*
    then show *?thesis* **by** (*meson a0 LocalRG-HoareDef.conseqPrePost subset-eq*)
   qed
  }
  then have *f3*:∀ *p-n*. Γ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,*Q p-n*,*Qa*]
   by *auto*
  then have ∀ *p-n*. Γ,Θ⊢$_{/F}$ *C sat* [{*s*. ∀ *p-n*. *s*∈*P p-n*}, *R*, *G*,{*s*. ∀ *p-n*. *s*∈*Q p-n*},*Qa*]
   using *all-Post* **by** *auto*
  moreover have *s1*:∀ *P*. {*s*. ∀ *p-n*. *s*∈*P p-n*} = (⋂ *p-n*. *P p-n*)
   by *auto*
  ultimately show *?thesis*
   by (*simp add*: *s1*)
qed

inductive-cases *hoare-elim-skip-cases* [*cases set*]:
$\Gamma,\Theta\vdash_{/F}$ *Skip sat* [p, R, G, q,a]

end

13 Derived Hoare Rules for Partial Correctness

theory *HoarePartial* **imports** *HoarePartialProps* **begin**

lemma *conseq-no-aux*:
$[\![\Gamma,\Theta\vdash_{/F} P'\ c\ Q',A';$
$\forall s.\ s \in P \longrightarrow (s{\in}P' \wedge (Q' \subseteq Q) \wedge (A' \subseteq A))]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
by (*rule conseq* [**where** $P'{=}\lambda Z.\ P'$ **and** $Q'{=}\lambda Z.\ Q'$ **and** $A'{=}\lambda Z.\ A'$]) *auto*

lemma *conseq-exploit-pre*:
$[\![\forall s \in P.\ \Gamma,\Theta\vdash_{/F} (\{s\} \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule Conseq*)
apply *clarify*
apply (*rule-tac* $x{=}\{s\} \cap P$ **in** *exI*)
apply (*rule-tac* $x{=}Q$ **in** *exI*)
apply (*rule-tac* $x{=}A$ **in** *exI*)
by *simp*

lemma *conseq*:$[\![\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\forall s.\ s \in P \longrightarrow (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
by (*rule Conseq'*) *blast*

lemma *Lem*: $[\![\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A)\}]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (lem\ x\ c)\ Q,A$
apply (*unfold lem-def*)
apply (*erule conseq*)

apply *blast*
done

lemma *LemAnno*:
assumes *conseq*: $P \subseteq \{s. \exists Z. s \in P' Z \wedge$
$(\forall t. t \in Q' Z \longrightarrow t \in Q) \wedge (\forall t. t \in A' Z \longrightarrow t \in A)\}$
assumes *lem*: $\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ (Q' Z),(A' Z)$
shows $\Gamma,\Theta\vdash_{/F} P\ (lem\ x\ c)\ Q,A$
apply (*rule Lem* [*OF lem*])
using *conseq*
by *blast*

lemma *LemAnnoNoAbrupt*:
assumes *conseq*: $P \subseteq \{s. \exists Z. s \in P' Z \wedge (\forall t. t \in Q' Z \longrightarrow t \in Q)\}$
assumes *lem*: $\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ (Q' Z),\{\}$
shows $\Gamma,\Theta\vdash_{/F} P\ (lem\ x\ c)\ Q,\{\}$
apply (*rule Lem* [*OF lem*])
using *conseq*
by *blast*

lemma *TrivPost*: $\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ (Q' Z),(A' Z)$
$\Longrightarrow$
$\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ UNIV,UNIV$
apply (*rule allI*)
apply (*erule conseq*)
apply *auto*
done

lemma *TrivPostNoAbr*: $\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ (Q' Z),\{\}$
$\Longrightarrow$
$\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ UNIV,\{\}$
apply (*rule allI*)
apply (*erule conseq*)
apply *auto*
done

lemma *conseq-under-new-pre*: $[\![\Gamma,\Theta\vdash_{/F} P'\ c\ Q',A';$
$\forall s \in P. s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule conseq*)
apply (*rule allI*)
apply *assumption*
apply *auto*
done

lemma *conseq-Kleymann*: $[\![\forall Z. \Gamma,\Theta\vdash_{/F} (P' Z)\ c\ (Q' Z),(A' Z);$
$\forall s \in P. (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A))]\!]$
$\Longrightarrow$

$\Gamma,\Theta\vdash_{/F}$ P c Q,A
by (*rule Conseq'*) *blast*

lemma *DynComConseq*:
assumes $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \ \Gamma,\Theta\vdash_{/F} P'\ (c\ s)\ Q',A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
shows $\Gamma,\Theta\vdash_{/F}$ P *DynCom* c Q,A
using *assms*
apply −
apply (*rule DynCom*)
apply *clarsimp*
apply (*rule Conseq*)
apply *clarsimp*
apply *blast*
done

lemma *SpecAnno*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))\}$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z)$
assumes *bdy-constant*: $\forall Z.\ c\ Z = c$ *undefined*
shows $\Gamma,\Theta\vdash_{/F}$ P (*specAnno* P' c Q' A') Q,A
proof −
from *spec bdy-constant*
have $\forall Z.\ \Gamma,\Theta\vdash_{/F} ((P'\ Z))$ (c *undefined*) $(Q'\ Z),(A'\ Z)$
apply −
apply (*rule allI*)
apply (*erule-tac x=Z* **in** *allE*)
apply (*erule-tac x=Z* **in** *allE*)
apply *simp*
done
with *consequence* **show** *?thesis*
apply (*simp add*: *specAnno-def*)
apply (*erule conseq*)
apply *blast*
done
qed

lemma *SpecAnno'*:
$[\![P \subseteq \{s.\ \ \exists\ Z.\ s{\in}P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ \ t \in Q) \wedge (\forall t.\ t \in A'\ Z \longrightarrow t \in\ \ A)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z);$
$\forall Z.\ c\ Z = c$ *undefined*
$]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ P (*specAnno* P' c Q' A') Q,A
apply (*simp only*: *subset-iff* [*THEN sym*])
apply (*erule* (*1*) *SpecAnno*)
apply *assumption*
done

lemma *SpecAnnoNoAbrupt*:
$[\![P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ (c\ Z)\ (Q'\ Z),\{\};$
$\forall Z.\ c\ Z = c\ \mathit{undefined}$
$]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{specAnno}\ P'\ c\ Q'\ (\lambda s.\ \{\}))\ Q,A$
apply (*rule SpecAnno'*)
apply *auto*
done

lemma *Skip*: $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ \mathit{Skip}\ Q,A$
by (*rule hoarep.Skip [THEN conseqPre],simp*)

lemma *Basic*: $P \subseteq \{s.\ (f\ s) \in Q\} \Longrightarrow\ \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Basic}\ f)\ Q,A$
by (*rule hoarep.Basic [THEN conseqPre]*)

lemma *BasicCond*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s{\in}Q) \wedge (\neg\ b\ s \longrightarrow g\ s{\in}Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ \mathit{Basic}\ (\lambda s.\ \mathit{if}\ b\ s\ \mathit{then}\ f\ s\ \mathit{else}\ g\ s)\ Q,A$
apply (*rule Basic*)
apply *auto*
done

lemma *Spec*: $P \subseteq \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Spec}\ r)\ Q,A$
by (*rule hoarep.Spec [THEN conseqPre]*)

lemma *SpecIf*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ \mathit{Spec}\ (\mathit{if\text{-}rel}\ b\ f\ g\ h)\ Q,A$
apply (*rule Spec*)
apply (*auto simp add: if-rel-def*)
done

lemma *Seq* [*trans, intro?*]:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Seq}\ c_1\ c_2)\ Q,A$
by (*rule hoarep.Seq*)

lemma *SeqSwap*:
$[\![\Gamma,\Theta\vdash_{/F}\ R\ \mathit{c2}\ Q,A;\ \Gamma,\Theta\vdash_{/F}\ P\ \mathit{c1}\ R,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Seq}\ \mathit{c1}\ \mathit{c2})\ Q,A$
by (*rule Seq*)

lemma *BSeq*:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{bseq}\ c_1\ c_2)\ Q,A$

by (*unfold bseq-def*) (*rule Seq*)

lemma *Cond*:
assumes *wp*: $P \subseteq \{s. (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
assumes *deriv-c1*: $\Gamma,\Theta\vdash_{/F} P_1\ c_1\ Q,A$
assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F} P_2\ c_2\ Q,A$
shows $\Gamma,\Theta\vdash_{/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$
proof (*rule hoarep.Cond* [*THEN conseqPre*])
from *deriv-c1*
show $\Gamma,\Theta\vdash_{/F} (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap b)\ c_1\ Q,A$
by (*rule conseqPre*) *blast*
next
from *deriv-c2*
show $\Gamma,\Theta\vdash_{/F} (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap -\ b)\ c_2\ Q,A$
by (*rule conseqPre*) *blast*
next
show $P \subseteq \{s. (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$ **by** (*rule wp*)
qed

lemma *CondSwap*:
$[\![\Gamma,\Theta\vdash_{/F} P1\ c1\ Q,A; \Gamma,\Theta\vdash_{/F} P2\ c2\ Q,A; P \subseteq \{s. (s{\in}b \longrightarrow s{\in}P1) \wedge (s{\notin}b \longrightarrow s{\in}P2)\}]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (Cond\ b\ c1\ c2)\ Q,A$
by (*rule Cond*)

lemma *Cond'*:
$[\![P \subseteq \{s. (b \subseteq P1) \wedge (-\ b \subseteq P2)\};\Gamma,\Theta\vdash_{/F} P1\ c1\ Q,A; \Gamma,\Theta\vdash_{/F} P2\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (Cond\ b\ c1\ c2)\ Q,A$
by (*rule CondSwap*) *blast+*

lemma *CondInv*:
assumes *wp*: $P \subseteq Q$
assumes *inv*: $Q \subseteq \{s. (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
assumes *deriv-c1*: $\Gamma,\Theta\vdash_{/F} P_1\ c_1\ Q,A$
assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F} P_2\ c_2\ Q,A$
shows $\Gamma,\Theta\vdash_{/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$
proof −
from *wp inv*
have $P \subseteq \{s. (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
by *blast*
from *Cond* [*OF this deriv-c1 deriv-c2*]
show *?thesis* **.**
qed

lemma *CondInv'*:
  assumes *wp*: $P \subseteq I$
  assumes *inv*: $I \subseteq \{s.\ (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
  assumes *wp'*: $I \subseteq Q$
  assumes *deriv-c1*: $\Gamma,\Theta\vdash_{/F}\ P_1\ c_1\ I,A$
  assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F}\ P_2\ c_2\ I,A$
  shows $\Gamma,\Theta\vdash_{/F}\ P\ (Cond\ b\ c_1\ c_2)\ Q,A$
proof −
  from *CondInv* [*OF wp inv deriv-c1 deriv-c2*]
  have $\Gamma,\Theta\vdash_{/F}\ P\ (Cond\ b\ c_1\ c_2)\ I,A$**.**
  from *conseqPost* [*OF this wp' subset-refl*]
  show *?thesis* **.**
qed

lemma *switchNil*:
  $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (switch\ v\ [])\ Q,A$
  by (*simp add*: *Skip*)

lemma *switchCons*:
  $[\![P \subseteq \{s.\ (v\ s \in V \longrightarrow s \in P_1) \wedge (v\ s \notin V \longrightarrow s \in P_2)\};$
    $\Gamma,\Theta\vdash_{/F}\ P_1\ c\ Q,A;$
    $\Gamma,\Theta\vdash_{/F}\ P_2\ (switch\ v\ vs)\ Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (switch\ v\ ((V,c)\#vs))\ Q,A$
  by (*simp add*: *Cond*)

lemma *Guard*:
$[\![P \subseteq g \cap R;\ \Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Guard\ f\ g\ c)\ Q,A$
apply (*rule Guard* [*THEN conseqPre*, *of - - - - R*])
apply (*erule conseqPre*)
apply *auto*
done

lemma *GuardSwap*:
$[\![\ \Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A;\ P \subseteq g \cap R]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Guard\ f\ g\ c)\ Q,A$
  by (*rule Guard*)

lemma *Guarantee*:
$[\![P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ \Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A;\ f \in F]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Guard\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*, *of - - - - -* $\{s.\ s \in g \longrightarrow s \in R\}$])
apply *assumption*
apply (*erule conseqPre*)
apply *auto*

done

lemma *GuaranteeSwap*:
$[\![$ $\Gamma,\Theta\vdash_{/F}$ R c Q,A; $P \subseteq \{s.\ s \in g \longrightarrow s \in R\}$; $f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(Guard\ f\ g\ c)$ Q,A
by (*rule Guarantee*)

lemma *GuardStrip*:
$[\![P \subseteq R$; $\Gamma,\Theta\vdash_{/F}$ R c Q,A; $f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(Guard\ f\ g\ c)$ Q,A
apply (*rule Guarantee* [*THEN conseqPre*])
apply *auto*
done

lemma *GuardStripSwap*:
$[\![\Gamma,\Theta\vdash_{/F}$ R c Q,A; $P \subseteq R$; $f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(Guard\ f\ g\ c)$ Q,A
by (*rule GuardStrip*)

lemma *GuaranteeStrip*:
$[\![P \subseteq R$; $\Gamma,\Theta\vdash_{/F}$ R c Q,A; $f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(guaranteeStrip\ f\ g\ c)$ Q,A
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeStripSwap*:
$[\![\Gamma,\Theta\vdash_{/F}$ R c Q,A; $P \subseteq R$; $f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(guaranteeStrip\ f\ g\ c)$ Q,A
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeAsGuard*:
$[\![P \subseteq g \cap R$; $\Gamma,\Theta\vdash_{/F}$ R c $Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(guaranteeStrip\ f\ g\ c)$ Q,A
by (*unfold guaranteeStrip-def*) (*rule Guard*)

lemma *GuaranteeAsGuardSwap*:
$[\![$ $\Gamma,\Theta\vdash_{/F}$ R c Q,A; $P \subseteq g \cap R]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}$ P $(guaranteeStrip\ f\ g\ c)$ Q,A
by (*rule GuaranteeAsGuard*)

lemma *GuardsNil*:
$\Gamma,\Theta\vdash_{/F}$ P c $Q,A \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ P $(guards\ []\ c)$ Q,A
by *simp*

lemma *GuardsCons*:
$\Gamma,\Theta\vdash_{/F}$ P $Guard\ f\ g$ $(guards\ gs\ c)$ $Q,A \Longrightarrow$

$\Gamma,\Theta\vdash_{/F}$ *P* (*guards* ((*f*,*g*)#*gs*) *c*) *Q*,*A*
by *simp*

lemma *GuardsConsGuaranteeStrip*:
$\Gamma,\Theta\vdash_{/F}$ *P guaranteeStrip f g* (*guards gs c*) *Q*,*A* $\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ *P* (*guards* (*guaranteeStripPair f g*#*gs*) *c*) *Q*,*A*
by (*simp add*: *guaranteeStripPair-def guaranteeStrip-def*)

lemma *While*:
assumes *P-I*: $P \subseteq I$
assumes *deriv-body*: $\Gamma,\Theta\vdash_{/F}$ ($I \cap b$) *c I*,*A*
assumes *I-Q*: $I \cap -b \subseteq Q$
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*whileAnno b I V c*) *Q*,*A*
proof −
from *deriv-body P-I I-Q*
show *?thesis*
apply (*simp add*: *whileAnno-def*)
apply (*erule conseqPrePost* [*OF HoarePartialDef.While*])
apply *simp-all*
done
qed

J will be instantiated by tactic with $gs' \cap I$ for those guards that are not stripped.

lemma *WhileAnnoG*:
$\Gamma,\Theta\vdash_{/F}$ *P* (*guards gs*
(*whileAnno b J V* (*Seq c* (*guards gs Skip*)))) *Q*,*A*
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}$ *P* (*whileAnnoG gs b I V c*) *Q*,*A*
by (*simp add*: *whileAnnoG-def whileAnno-def while-def*)

This form stems from *strip-guards F* (*whileAnnoG gs b I V c*)

lemma *WhileNoGuard'*:
assumes *P-I*: $P \subseteq I$
assumes *deriv-body*: $\Gamma,\Theta\vdash_{/F}$ ($I \cap b$) *c I*,*A*
assumes *I-Q*: $I \cap -b \subseteq Q$
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*whileAnno b I V* (*Seq c Skip*)) *Q*,*A*
apply (*rule While* [*OF P-I - I-Q*])
apply (*rule Seq*)
apply (*rule deriv-body*)
apply (*rule hoarep.Skip*)
done

lemma *WhileAnnoFix*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s \in I\ Z \wedge (I\ Z \cap -b \subseteq Q))\ \}$
assumes *bdy*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ ($I\ Z \cap b$) ($c\ Z$) ($I\ Z$),*A*
assumes *bdy-constant*: $\forall Z.\ c\ Z = c$ *undefined*

shows $\Gamma,\Theta\vdash_{/F}$ P (*whileAnnoFix b I V c*) Q,A
proof −
  from *bdy bdy-constant*
  have *bdy'*: $\forall Z$. $\Gamma,\Theta\vdash_{/F}$ (I Z $\cap$ b) (*c undefined*) (I Z),A
    apply −
    apply (*rule allI*)
    apply (*erule-tac x=Z* **in** *allE*)
    apply (*erule-tac x=Z* **in** *allE*)
    apply *simp*
    done
  have $\forall Z$. $\Gamma,\Theta\vdash_{/F}$ (I Z) (*whileAnnoFix b I V c*) (I Z $\cap$ $-b$),A
    apply *rule*
    apply (*unfold whileAnnoFix-def*)
    apply (*rule hoarep.While*)
    apply (*rule bdy'* [*rule-format*])
    done
  then
  show *?thesis*
    apply (*rule conseq*)
    using *consequence*
    by *blast*
qed

lemma *WhileAnnoFix'*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}I\ Z\ \wedge$
$(\forall t.\ t \in I\ Z \cap -b \longrightarrow t \in Q))\ \}$
assumes *bdy*: $\forall Z$. $\Gamma,\Theta\vdash_{/F}$ (I Z $\cap$ b) (c Z) (I Z),A
assumes *bdy-constant*: $\forall Z.\ c\ Z = c$ *undefined*
shows $\Gamma,\Theta\vdash_{/F}$ P (*whileAnnoFix b I V c*) Q,A
  apply (*rule WhileAnnoFix* [*OF - bdy bdy-constant*])
  using *consequence* **by** *blast*

lemma *WhileAnnoGFix*:
assumes *whileAnnoFix*:
  $\Gamma,\Theta\vdash_{/F}$ P (*guards gs*
    (*whileAnnoFix b J V* (λZ. (*Seq* (c Z) (*guards gs Skip*))))) Q,A
shows $\Gamma,\Theta\vdash_{/F}$ P (*whileAnnoGFix gs b I V c*) Q,A
  using *whileAnnoFix*
  by (*simp add*: *whileAnnoGFix-def whileAnnoFix-def while-def*)

lemma *Bind*:
  assumes *adapt*: $P \subseteq \{s.\ s \in P'\ s\}$
  assumes *c*: $\forall s$. $\Gamma,\Theta\vdash_{/F}$ (P' s) (c (e s)) Q,A
  shows $\Gamma,\Theta\vdash_{/F}$ P (*bind e c*) Q,A
apply (*rule conseq* [**where** $P'{=}\lambda Z.\ \{s.\ s{=}Z \wedge s \in P'\ Z\}$ **and** $Q'{=}\lambda Z.\ Q$ **and** $A'{=}\lambda Z.\ A$])
apply (*rule allI*)
apply (*unfold bind-def*)

apply (*rule DynCom*)
apply (*rule ballI*)
apply *simp*
apply (*rule conseqPre*)
apply (*rule c* [*rule-format*])
apply *blast*
using *adapt*
apply *blast*
done

lemma *Block*:
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{/F}\ (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge init\ s \in P'\ Z\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in R\ Z\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ Z\ t \in R\ Z\ t\}$ **and**
$Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule DynCom*)
apply (*clarsimp*)
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in A\}$ **in** *Catch*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule bdy* [*rule-format*])
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *BlockSwap*:
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{/F}\ (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (block\ init\ bdy\ return\ c)\ Q,A$
using *adapt bdy c*
 by (*rule Block*)

lemma *BlockSpec*:
 assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
 $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
 $(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
 assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
 assumes *bdy*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ bdy\ (Q'\ Z),(A'\ Z)$
 shows $\Gamma,\Theta\vdash_{/F}\ P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ init\ s \in P'\ Z\ \wedge$
 $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
 $(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ s\ t \in R\ s\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ s\ t \in R\ s\ t\}$ **and**
 $Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule DynCom*)
apply (*clarsimp*)
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ s\ t \in A\}$ **in** *Catch*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule conseq* [*OF bdy*])
apply *clarsimp*
apply *blast*

apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *Throw*: $P \subseteq A \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ Throw\ Q,A$
by (*rule hoarep.Throw* [*THEN conseqPre*])

lemmas *Catch* = *hoarep.Catch*
lemma *CatchSwap*: $[\![\Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A;\ \Gamma,\Theta\vdash_{/F} P\ c_1\ Q,R]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ Catch\ c_1\ c_2\ Q,A$
by (*rule hoarep.Catch*)

lemma *raise*: $P \subseteq \{s.\ f\ s \in A\} \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ raise\ f\ Q,A$
apply (*simp add*: *raise-def*)
apply (*rule Seq*)
apply (*rule Basic*)
apply (*assumption*)
apply (*rule Throw*)
apply (*rule subset-refl*)
done

lemma *condCatch*: $[\![\Gamma,\Theta\vdash_{/F} P\ c_1\ Q,((b \cap R) \cup (-b \cap A));\Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A]\!]$
$\Longrightarrow\ \Gamma,\Theta\vdash_{/F}P\ condCatch\ c_1\ b\ c_2\ Q,A$
apply (*simp add*: *condCatch-def*)
apply (*rule Catch*)
apply *assumption*
apply (*rule CondSwap*)
apply (*assumption*)
apply (*rule hoarep.Throw*)
apply *blast*
done

lemma *condCatchSwap*: $[\![\Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A;\Gamma,\Theta\vdash_{/F} P\ c_1\ Q,((b \cap R) \cup (-b \cap A))]\!]$
$\Longrightarrow\ \Gamma,\Theta\vdash_{/F}P\ condCatch\ c_1\ b\ c_2\ Q,A$
by (*rule condCatch*)

lemma *ProcSpec*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
$(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ Call\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F} P\ (call\ init\ p\ return\ c)\ Q,A$
using *adapt c p*

apply (*unfold call-def*)
by (*rule BlockSpec*)

lemma *ProcSpec'*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z \wedge$
$(\forall t \in Q'\ Z.\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ Z.\ return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
apply (*insert adapt*)
apply *clarsimp*
apply (*drule* (*1*) *subsetD*)
apply (*clarsimp*)
apply (*rule-tac x=Z* **in** *exI*)
apply *blast*
done

lemma *ProcSpecNoAbrupt*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ p\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
using *adapt*
apply *simp*
done

lemma *FCall*:
$\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ (\lambda s\ t.\ c\ (result\ t)))\ Q,A$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (fcall\ init\ p\ return\ result\ c)\ Q,A$
by (*simp add*: *fcall-def*)

lemma *ProcRec*:
assumes *deriv-bodies*:
$\forall p{\in}Procs.$
$\forall Z.\ \Gamma,\Theta\cup(\bigcup p{\in}Procs.\ \bigcup Z.\ \{(P\ p\ Z,p,Q\ p\ Z,A\ p\ Z)\})$
$\vdash_{/F}\ (P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
assumes *Procs-defined*: $Procs \subseteq dom\ \Gamma$
shows $\forall p{\in}Procs.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}(P\ p\ Z)\ Call\ p\ (Q\ p\ Z),(A\ p\ Z)$
by (*intro strip*)
(*rule CallRec'*
[*OF - Procs-defined deriv-bodies*],
simp-all)

lemma *ProcRec′*:
  assumes *ctxt*: $\Theta' = \Theta\cup(\bigcup p\in Procs.\ \bigcup Z.\ \{(P\ p\ Z,p,Q\ p\ Z,A\ p\ Z)\})$
  assumes *deriv-bodies*:
   $\forall p\in Procs.\ \forall Z.\ \Gamma,\Theta'\vdash_{/F}\ (P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
  assumes *Procs-defined*: $Procs \subseteq dom\ \Gamma$
  shows $\forall p\in Procs.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}(P\ p\ Z)\ Call\ p\ (Q\ p\ Z),(A\ p\ Z)$
  using *ctxt deriv-bodies*
  apply *simp*
  apply (*erule ProcRec* [*OF - Procs-defined*])
  done

lemma *ProcRecList*:
  assumes *deriv-bodies*:
   $\forall p\in set\ Procs.$
    $\forall Z.\ \Gamma,\Theta\cup(\bigcup p\in set\ Procs.\ \bigcup Z.\ \{(P\ p\ Z,p,Q\ p\ Z,A\ p\ Z)\})$
      $\vdash_{/F}\ (P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
  assumes *dist*: *distinct Procs*
  assumes *Procs-defined*: $set\ Procs \subseteq dom\ \Gamma$
  shows $\forall p\in set\ Procs.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}(P\ p\ Z)\ Call\ p\ (Q\ p\ Z),(A\ p\ Z)$
  using *deriv-bodies Procs-defined*
  by (*rule ProcRec*)

lemma *ProcRecSpecs*:
  $[\![\forall (P,p,Q,A) \in Specs.\ \Gamma,\Theta\cup Specs\vdash_{/F}\ P\ (the\ (\Gamma\ p))\ Q,A;$
   $\forall (P,p,Q,A) \in Specs.\ p \in dom\ \Gamma]\!]$
  $\Longrightarrow \forall (P,p,Q,A) \in Specs.\ \Gamma,\Theta\vdash_{/F}\ P\ (Call\ p)\ Q,A$
apply (*auto intro*: *CallRec*)
done

lemma *ProcRec1*:
  assumes *deriv-body*:
   $\forall Z.\ \Gamma,\Theta\cup(\bigcup Z.\ \{(P\ Z,p,Q\ Z,A\ Z)\})\vdash_{/F}\ (P\ Z)\ (the\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
  assumes *p-defined*: $p \in dom\ \Gamma$
  shows $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P\ Z)\ Call\ p\ (Q\ Z),(A\ Z)$
proof –
  from *deriv-body p-defined*
  have $\forall p\in\{p\}.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P\ Z)\ Call\ p\ (Q\ Z),(A\ Z)$
    by – (*rule ProcRec* [**where** $A=\lambda p.\ A$ **and** $P=\lambda p.\ P$ **and** $Q=\lambda p.\ Q$],
      simp-all)
  thus *?thesis*
    by *simp*
qed

lemma *ProcNoRec1*:
  assumes *deriv-body*:
   $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P\ Z)\ (the\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$

assumes *p-def*: $p \in dom\ \Gamma$
shows $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ Call\ p\ (Q\ Z),(A\ Z)$
proof −
from *deriv-body*
have $\forall Z.\ \Gamma,\Theta\cup(\bigcup Z.\ \{(P\ Z,p,Q\ Z,A\ Z)\})$
$\vdash_{/F} (P\ Z)\ (the\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
by (*blast intro*: *hoare-augment-context*)
from *this p-def*
show *?thesis*
by (*rule ProcRec1*)
qed

lemma *ProcBody*:
assumes *WP*: $P \subseteq P'$
assumes *deriv-body*: $\Gamma,\Theta\vdash_{/F} P'\ body\ Q,A$
assumes *body*: $\Gamma\ p = Some\ body$
shows $\Gamma,\Theta\vdash_{/F} P\ Call\ p\ Q,A$
apply (*rule conseqPre* [*OF - WP*])
apply (*rule ProcNoRec1* [*rule-format*, **where** $P=\lambda Z.\ P'$ **and** $Q=\lambda Z.\ Q$ **and** $A=\lambda Z.\ A$])
apply (*insert body*)
apply *simp*
apply (*rule hoare-augment-context* [*OF deriv-body*])
apply *blast*
apply *fastforce*
done

lemma *CallBody*:
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{/F} (P'\ s)\ body\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *body*: $\Gamma\ p = Some\ body$
shows $\Gamma,\Theta\vdash_{/F} P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*unfold call-def*)
apply (*rule Block* [*OF adapt - c*])
apply (*rule allI*)
apply (*rule ProcBody* [**where** $\Gamma=\Gamma$, *OF - bdy* [*rule-format*] *body*])
apply *simp*
done

lemmas *ProcModifyReturn* = *HoarePartialProps.ProcModifyReturn*
lemmas *ProcModifyReturnSameFaults* = *HoarePartialProps.ProcModifyReturnSameFaults*

lemma *ProcModifyReturnNoAbr*:
assumes *spec*: $\Gamma,\Theta\vdash_{/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:

$\forall\sigma.\ \Gamma,\Theta\vdash_{/UNIV}\ \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturn* [*OF spec result-conform - modifies-spec*]) *simp*

lemma *ProcModifyReturnNoAbrSameFaults*:
assumes *spec*: $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall\sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturnSameFaults* [*OF spec result-conform - modifies-spec*]) *simp*

lemma *DynProc*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ s\ Z \wedge$
$(\forall t.\ t \in Q'\ s\ Z \longrightarrow\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ s\ Z \longrightarrow return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s{\in}\ P.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ s\ Z)\ Call\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ dynCall\ init\ p\ return\ c\ Q,A$
apply (*rule conseq* [**where** $P'{=}\lambda Z.\ \{s.\ s{=}Z \wedge s \in P\}$
and $Q'{=}\lambda Z.\ Q$ **and** $A'{=}\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold dynCall-def call-def block-def*)
apply (*rule DynCom*)
apply *clarsimp*
apply (*rule DynCom*)
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])
apply *clarsimp*
apply (*rename-tac Z'*)
apply (*rule-tac* $R{=}Q'\ Z\ Z'$ **in** *Seq*)
apply (*rule CatchSwap*)
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule subset-refl*)
apply (*rule Basic*)
apply (*rule subset-refl*)
apply (*rule-tac* $R{=}\{i.\ i \in P'\ Z\ Z'\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule-tac* $Q'{=}Q'\ Z\ Z'$ **and** $A'{=}A'\ Z\ Z'$ **in** *conseqPost*)

using *p*
apply *clarsimp*
apply *simp*
apply *clarsimp*
apply (*rule-tac* $P' = \lambda Z''. \{t. t = Z'' \wedge return\ Z\ t \in R\ Z\ t\}$ **and**
$Q' = \lambda Z''. Q$ **and** $A' = \lambda Z''. A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule DynCom*)
apply *clarsimp*
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
done

lemma *DynProc'*:
assumes *adapt*: $P \subseteq \{s. \exists Z. init\ s \in P'\ s\ Z \wedge$
$(\forall t \in Q'\ s\ Z. return\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ s\ Z. return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t. \Gamma,\Theta \vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s \in P. \forall Z. \Gamma,\Theta \vdash_{/F} (P'\ s\ Z)\ Call\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta \vdash_{/F} P\ dynCall\ init\ p\ return\ c\ Q,A$
proof –
from *adapt* **have** $P \subseteq \{s. \exists Z. init\ s \in P'\ s\ Z \wedge$
$(\forall t. t \in Q'\ s\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
$(\forall t. t \in A'\ s\ Z \longrightarrow return\ s\ t \in A)\}$
by *blast*
from *this c p* **show** *?thesis*
by (*rule DynProc*)
qed

lemma *DynProcStaticSpec*:
assumes *adapt*: $P \subseteq \{s. s \in S \wedge (\exists Z. init\ s \in P'\ Z \wedge$
$(\forall \tau. \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau. \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t. \Gamma,\Theta \vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall s \in S. \forall Z. \Gamma,\Theta \vdash_{/F} (P'\ Z)\ Call\ (p\ s)\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta \vdash_{/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof –
from *adapt* **have** *P-S*: $P \subseteq S$
by *blast*
have $\Gamma,\Theta \vdash_{/F} (P \cap S)\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProc* [**where** $P' = \lambda s\ Z. P'\ Z$ **and** $Q' = \lambda s\ Z. Q'\ Z$
and $A' = \lambda s\ Z. A'\ Z$, *OF - c*])
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])

apply *clarsimp*
using *spec*
apply *clarsimp*
done
thus *?thesis*
by (*rule conseqPre*) (*insert P-S*,*blast*)
qed

lemma *DynProcProcPar*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ q\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProcStaticSpec* [**where** $S=\{s.\ p\ s = q\}$,*simplified*, *OF adapt c*])
using *spec*
apply *simp*
done

lemma *DynProcProcParNoAbrupt*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ q\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof −
have $P \subseteq \{s.\ p\ s = q \wedge (\exists\ Z.\ init\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in \{\} \longrightarrow return\ s\ t \in A))\}$
(**is** $P \subseteq ?P'$)
proof
fix *s*
assume *P*: $s \in P$
with *adapt* **obtain** *Z* **where**
Pre: $p\ s = q \wedge init\ s \in P'\ Z$ **and**
adapt-Norm: $\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau$
by *blast*
from *adapt-Norm*
have $\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t$
by *auto*
then
show $s \in ?P'$
using *Pre* **by** *blast*
qed
note $P = this$

show *?thesis*
  apply −
  apply (*rule DynProcStaticSpec* [**where** $S=\{s.\ p\ s = q\}$,*simplified*, *OF P c*])
  apply (*insert spec*)
  apply *auto*
  done
qed

lemma *DynProcModifyReturnNoAbr*:
  assumes *to-prove*: $\Gamma,\Theta\vdash_{/F} P$ (*dynCall init p return′ c*) Q,A
  assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  assumes *modif-clause*:
  $\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ Call\ (p\ s)\ \ (Modif\ \sigma),\{\}$
  shows $\Gamma,\Theta\vdash_{/F} P$ (*dynCall init p return c*) Q,A
proof −
  from *ret-nrm-modif*
  have $\forall s\ t.\ t\ \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  by *iprover*
  then
  have *ret-nrm-modif′*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  by *simp*
  have *ret-abr-modif′*: $\forall s\ t.\ t \in \{\}$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  by *simp*
  from *to-prove ret-nrm-modif′ ret-abr-modif′ modif-clause* **show** *?thesis*
  by (*rule dynProcModifyReturn*)
qed

lemma *ProcDynModifyReturnNoAbrSameFaults*:
  assumes *to-prove*: $\Gamma,\Theta\vdash_{/F} P$ (*dynCall init p return′ c*) Q,A
  assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  assumes *modif-clause*:
  $\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),\{\}$
  shows $\Gamma,\Theta\vdash_{/F} P$ (*dynCall init p return c*) Q,A
proof −
  from *ret-nrm-modif*
  have $\forall s\ t.\ t\ \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  by *iprover*
  then
  have *ret-nrm-modif′*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$

by *simp*
have *ret-abr-modif'*: $\forall s\ t.\ t \in \{\}$
$\longrightarrow$ *return'* $s\ t$ = *return* $s\ t$
by *simp*
from *to-prove ret-nrm-modif' ret-abr-modif' modif-clause* **show** *?thesis*
by (*rule dynProcModifyReturnSameFaults*)
qed

lemma *ProcProcParModifyReturn*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcPar* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in$ (*Modif* (*init s*))
$\longrightarrow$ *return'* $s\ t$ = *return* $s\ t$
assumes *ret-abr-modif*: $\forall s\ t.\ t \in$ (*ModifAbr* (*init s*))
$\longrightarrow$ *return'* $s\ t$ = *return* $s\ t$
assumes *modif-clause*:
$\forall\sigma.\ \Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ (*Call q*) (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{/F}$ P (*dynCall init p return c*) Q,A
proof −
from *to-prove* **have** $\Gamma,\Theta\vdash_{/F}$ $(\{s.\ p\ s = q\} \cap P')$ (*dynCall init p return' c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
ret-abr-modif
have $\Gamma,\Theta\vdash_{/F}$ $(\{s.\ p\ s = q\} \cap P')$ (*dynCall init p return c*) Q,A
by (*rule dynProcModifyReturn*) (*insert modif-clause*,*auto*)
from *this q* **show** *?thesis*
by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnSameFaults*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcPar* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in$ (*Modif* (*init s*))
$\longrightarrow$ *return'* $s\ t$ = *return* $s\ t$
assumes *ret-abr-modif*: $\forall s\ t.\ t \in$ (*ModifAbr* (*init s*))
$\longrightarrow$ *return'* $s\ t$ = *return* $s\ t$
assumes *modif-clause*:
$\forall\sigma.\ \Gamma,\Theta\vdash_{/F}$ $\{\sigma\}$ *Call q* (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{/F}$ P (*dynCall init p return c*) Q,A
proof −
from *to-prove*
have $\Gamma,\Theta\vdash_{/F}$ $(\{s.\ p\ s = q\} \cap P')$ (*dynCall init p return' c*) Q,A

by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
ret-abr-modif
have $\Gamma,\Theta\vdash_{/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
by (*rule dynProcModifyReturnSameFaults*) (*insert modif-clause*,*auto*)
from *this q* **show** *?thesis*
by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnNoAbr*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{/F}$ P (*dynCall init p return c*) Q,A
proof −
from *to-prove* **have** $\Gamma,\Theta\vdash_{/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
have $\Gamma,\Theta\vdash_{/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
by (*rule DynProcModifyReturnNoAbr*) (*insert modif-clause*,*auto*)
from *this q* **show** *?thesis*
by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnNoAbrSameFaults*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{/F}$ P (*dynCall init p return c*) Q,A
proof −
from *to-prove* **have**
$\Gamma,\Theta\vdash_{/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
have $\Gamma,\Theta\vdash_{/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
by (*rule ProcDynModifyReturnNoAbrSameFaults*) (*insert modif-clause*,*auto*)

from *this q* **show** *?thesis*
  by (*rule conseqPre*)
qed

lemma *MergeGuards-iff*: $\Gamma,\Theta\vdash_{/F}$ *P merge-guards c Q,A* = $\Gamma,\Theta\vdash_{/F}$ *P c Q,A*
  by (*auto intro*: *MergeGuardsI MergeGuardsD*)

lemma *CombineStrip'*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ *P c' Q,A*
  assumes *deriv-strip-triv*: $\Gamma,\{\}\vdash_{/\{\}}$ *P c'' UNIV,UNIV*
  assumes *c''*: *c''= mark-guards False* (*strip-guards* (−*F*) *c'*)
  assumes *c*: *merge-guards c* = *merge-guards* (*mark-guards False c'*)
  shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q,A*
proof −
  from *deriv-strip-triv* **have** *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}$ *P c'' UNIV,UNIV*
    by (*auto intro*: *hoare-augment-context*)
  from *deriv-strip* [*simplified c''*]
  have $\Gamma,\Theta\vdash_{/\{\}}$ *P* (*strip-guards* (− *F*) *c'*) *UNIV,UNIV*
    by (*rule MarkGuardsD*)
  with *deriv*
  have $\Gamma,\Theta\vdash_{/\{\}}$ *P c' Q,A*
    by (*rule CombineStrip*)
  hence $\Gamma,\Theta\vdash_{/\{\}}$ *P mark-guards False c' Q,A*
    by (*rule MarkGuardsI*)
  hence $\Gamma,\Theta\vdash_{/\{\}}$ *P merge-guards* (*mark-guards False c'*) *Q,A*
    by (*rule MergeGuardsI*)
  hence $\Gamma,\Theta\vdash_{/\{\}}$ *P merge-guards c Q,A*
    by (*simp add*: *c*)
  thus *?thesis*
    by (*rule MergeGuardsD*)
qed

lemma *CombineStrip''*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/\{True\}}$ *P c' Q,A*
  assumes *deriv-strip-triv*: $\Gamma,\{\}\vdash_{/\{\}}$ *P c'' UNIV,UNIV*
  assumes *c''*: *c''= mark-guards False* (*strip-guards* ({*False*}) *c'*)
  assumes *c*: *merge-guards c* = *merge-guards* (*mark-guards False c'*)
  shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q,A*
  apply (*rule CombineStrip'* [*OF deriv deriv-strip-triv - c*])
  apply (*insert c''*)
  apply (*subgoal-tac* − {*True*} = {*False*})
  apply *auto*
  done

lemma *AsmUN*:
  ($\bigcup Z.$ {(*P Z*, *p*, *Q Z*,*A Z*)}) $\subseteq \Theta$
  $\Longrightarrow$

$\forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ (Call\ p)\ (Q\ Z),(A\ Z)$
by (*blast intro*: *hoarep.Asm*)

lemma *augment-context'*:
$[\![\Theta \subseteq \Theta'; \forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ \ p\ (Q\ Z),(A\ Z)]\!]$
$\Longrightarrow \forall Z.\ \Gamma,\Theta'\vdash_{/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
by (*iprover intro*: *hoare-augment-context*)

lemma *hoarep-strip*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{/F} (P\ Z)\ p\ (Q\ Z),(A\ Z);\ F' \subseteq -F]\!] \Longrightarrow$
$\forall Z.\ strip\ F'\ \Gamma,\{\}\vdash_{/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
by (*iprover intro*: *hoare-strip-*Γ)

lemma *augment-emptyFaults*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{/\{\}} (P\ Z)\ p\ (Q\ Z),(A\ Z)]\!] \Longrightarrow$
$\forall Z.\ \Gamma,\{\}\vdash_{/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
by (*blast intro*: *augment-Faults*)

lemma *augment-FaultsUNIV*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)]\!] \Longrightarrow$
$\forall Z.\ \Gamma,\{\}\vdash_{/UNIV} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
by (*blast intro*: *augment-Faults*)

lemma *PostConjI* [*trans*]:
$[\![\Gamma,\Theta\vdash_{/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{/F} P\ c\ R,B]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ (Q \cap R),(A \cap B)$
by (*rule PostConjI*)

lemma *PostConjI'* :
$[\![\Gamma,\Theta\vdash_{/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{/F} P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ R,B]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ (Q \cap R),(A \cap B)$
by (*rule PostConjI*) *iprover+*

lemma *PostConjE* [*consumes 1*]:
assumes *conj*: $\Gamma,\Theta\vdash_{/F} P\ c\ (Q \cap R),(A \cap B)$
assumes *E*: $[\![\Gamma,\Theta\vdash_{/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{/F} P\ c\ R,B]\!] \Longrightarrow S$
shows *S*
proof −
from *conj* **have** $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$ **by** (*rule conseqPost*) *blast+*
moreover
from *conj* **have** $\Gamma,\Theta\vdash_{/F} P\ c\ R,B$ **by** (*rule conseqPost*) *blast+*
ultimately show *S*
by (*rule E*)
qed

13.1 Rules for Single-Step Proof

We are now ready to introduce a set of Hoare rules to be used in single-step structured proofs in Isabelle/Isar.

Assertions of Hoare Logic may be manipulated in calculational proofs, with the inclusion expressed in terms of sets or predicates. Reversed order is supported as well.

lemma *annotateI* [*trans*]:
$[\![\Gamma,\Theta\vdash_{/F}P\ anno\ Q,A;\ c = anno]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ Q,A$
  by *simp*

lemma *annotate-normI*:
  assumes *deriv-anno*: $\Gamma,\Theta\vdash_{/F}P\ anno\ Q,A$
  assumes *norm-eq*: *normalize c = normalize anno*
  shows $\Gamma,\Theta\vdash_{/F}P\ c\ Q,A$
proof −
  from *NormalizeI* [*OF deriv-anno*] *norm-eq*
  have $\Gamma,\Theta\vdash_{/F}\ P\ normalize\ c\ Q,A$
    by *simp*
  from *NormalizeD* [*OF this*]
  show *?thesis* **.**
qed

lemma *annotateWhile*:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ (whileAnnoG\ gs\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (while\ gs\ b\ c)\ Q,A$
  by (*simp add*: *whileAnnoG-def*)

lemma *reannotateWhile*:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ (whileAnnoG\ gs\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (whileAnnoG\ gs\ b\ J\ V\ c)\ Q,A$
  by (*simp add*: *whileAnnoG-def*)

lemma *reannotateWhileNoGuard*:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ J\ V\ c)\ Q,A$
  by (*simp add*: *whileAnno-def*)

lemma [*trans*] : $P' \subseteq P \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P'\ c\ Q,A$
  by (*rule conseqPre*)

lemma [*trans*]: $Q \subseteq Q' \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q',A$
  by (*rule conseqPost*) *blast+*

lemma [*trans*]:
    $\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ c\ Q,A \Longrightarrow (\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P'\ s\}\ c\ Q,A$
  by (*rule conseqPre*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P'\ s\}\ c\ Q,A$
by (*rule conseqPre*) *auto*

lemma [*trans*]:
$\Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q\ s\},A \Longrightarrow (\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q\ s\},A \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*intro?*]: $\Gamma,\Theta\vdash_{/F}\ P\ Skip\ P,A$
by (*rule Skip*) *auto*

lemma *CondInt* [*trans*,*intro?*]:
$[\![\Gamma,\Theta\vdash_{/F}\ (P \cap b)\ c1\ Q,A;\ \Gamma,\Theta\vdash_{/F}\ (P \cap -\ b)\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (Cond\ b\ c1\ c2)\ Q,A$
by (*rule Cond*) *auto*

lemma *CondConj* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s \wedge b\ s\}\ c1\ Q,A;\ \Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s \wedge \neg\ b\ s\}\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ (Cond\ \{s.\ b\ s\}\ c1\ c2)\ Q,A$
by (*rule Cond*) *auto*

lemma *WhileInvInt* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}\ (P \cap b)\ c\ P,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ P\ V\ c)\ (P \cap -b),A$
by (*rule While*) *auto*

lemma *WhileInt* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}\ (P \cap b)\ c\ P,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ \{s.\ undefined\}\ V\ c)\ (P \cap -b),A$
by (*unfold whileAnno-def*)
(*rule HoarePartialDef.While* [*THEN conseqPrePost*],*auto*)

lemma *WhileInvConj* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s \wedge b\ s\}\ c\ \{s.\ P\ s\},A$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ (whileAnno\ \{s.\ b\ s\}\ \{s.\ P\ s\}\ V\ c)\ \{s.\ P\ s \wedge \neg\ b\ s\},A$
by (*simp add*: *While Collect-conj-eq Collect-neg-eq*)

lemma *WhileConj* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s \wedge b\ s\}\ c\ \{s.\ P\ s\},A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ (whileAnno\ \{s.\ b\ s\}\ \{s.\ undefined\}\ V\ c)\ \{s.\ P\ s \wedge \neg\ b\ s\},A$

by (*unfold whileAnno-def*)
(*simp add*: *HoarePartialDef.While* [*THEN conseqPrePost*]
Collect-conj-eq Collect-neg-eq)

end

14 Hoare Logic for Total Correctness

theory *HoareTotalDef* **imports** *HoarePartialDef Termination* **begin**

14.1 Validity of Hoare Tuples: $\Gamma\models_{t/F}$ *P c Q*,*A*

definition
validt :: [(*'s*,*'p*,*'f*) *body*,*'f set*,*'s assn*,(*'s*,*'p*,*'f*) *com*,*'s assn*,*'s assn*] ⇒ *bool*
($-\models_{t'/\_}/$ - - -,- [*61*,*60*,*1000*, *20*, *1000*,*1000*] *60*)
where
$\Gamma\models_{t/F}$ *P c Q*,*A* ≡ $\Gamma\models_{/F}$ *P c Q*,*A* ∧ (∀ *s* ∈ *Normal* ‘ *P*. Γ⊢*c*↓*s*)

definition
cvalidt::
[(*'s*,*'p*,*'f*) *body*,(*'s*,*'p*) *quadruple set*,*'f set*,
's assn,(*'s*,*'p*,*'f*) *com*,*'s assn*,*'s assn*] ⇒ *bool*
($-,-\models_{t'/\_}/$ - - -,- [*61*,*60*, *60*,*1000*, *20*, *1000*,*1000*] *60*)
where
Γ,Θ$\models_{t/F}$ *P c Q*,*A* ≡ (∀ (*P*,*p*,*Q*,*A*)∈Θ. Γ$\models_{t/F}$ *P* (*Call p*) *Q*,*A*) ⟶ Γ $\models_{t/F}$ *P c*
Q,*A*

notation (*ASCII*)
validt (-|=*t'*/-/ - - -,- [*61*,*60*,*1000*, *20*, *1000*,*1000*] *60*) **and**
cvalidt (-,-|=*t'*/- / - - -,- [*61*,*60*,*60*,*1000*, *20*, *1000*,*1000*] *60*)

14.2 Properties of Validity

lemma *validtI*:
⟦⋀*s t*. ⟦Γ⊢⟨*c*,*Normal s*⟩ ⇒ *t*;*s* ∈ *P*;*t* ∉ *Fault* ‘ *F*⟧ ⟹ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt*
‘ *A*;
⋀*s*. *s* ∈ *P* ⟹ Γ⊢ *c*↓(*Normal s*) ⟧
⟹ Γ$\models_{t/F}$ *P c Q*,*A*
by (*auto simp add*: *validt-def valid-def*)

lemma *cvalidtI*:
⟦⋀*s t*. ⟦∀ (*P*,*p*,*Q*,*A*)∈Θ. Γ$\models_{t/F}$ *P* (*Call p*) *Q*,*A*;Γ⊢⟨*c*,*Normal s*⟩ ⇒ *t*;*s* ∈ *P*;
t ∉ *Fault* ‘ *F*⟧
⟹ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*;

$\bigwedge s.\ [\![\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A;\ s\in P]\!] \Longrightarrow\ \Gamma\vdash c\downarrow(Normal\ s)]\!]$
$\Longrightarrow \Gamma,\Theta\models_{t/F}\ P\ c\ Q,A$
by (*auto simp add*: *cvalidt-def validt-def valid-def*)

lemma *cvalidt-postD*:
$[\![\Gamma,\Theta\models_{t/F}\ P\ c\ Q,A;\ \forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A;\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow t;$
$s \in P;t \notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A$
by (*simp add*: *cvalidt-def validt-def valid-def*)

lemma *cvalidt-termD*:
$[\![\Gamma,\Theta\models_{t/F}\ P\ c\ Q,A;\ \forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A;s \in P]\!]$
$\Longrightarrow \Gamma\vdash c\downarrow(Normal\ s)$
by (*simp add*: *cvalidt-def validt-def valid-def*)

lemma *validt-augment-Faults*:
assumes *valid*:$\Gamma\models_{t/F}\ P\ c\ Q,A$
assumes F': $F \subseteq F'$
shows $\Gamma\models_{t/F'}\ P\ c\ Q,A$
using *valid* F'
by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)

14.3 The Hoare Rules: $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$

inductive *hoaret*::[('s,'p,'f) body,('s,'p) quadruple set,'f set,
'*s assn*,('s,'p,'f) *com*,'*s assn*,'*s assn*]
=> *bool*
((3-,-/$\vdash_{t'/-}$ (-/ (-)/ -,-)) [61,60,60,1000,20,1000,1000]60)
for Γ::('s,'p,'f) *body*
where
Skip: $\Gamma,\Theta\vdash_{t/F}\ Q\ Skip\ Q,A$

| *Basic*: $\Gamma,\Theta\vdash_{t/F}\ \{s.\ f\ s \in Q\}\ (Basic\ f)\ Q,A$

| *Spec*: $\Gamma,\Theta\vdash_{t/F}\ \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}\ (Spec\ r)\ Q,A$

| *Seq*: $[\![\Gamma,\Theta\vdash_{t/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ Seq\ c_1\ c_2\ Q,A$

| *Cond*: $[\![\Gamma,\Theta\vdash_{t/F}\ (P \cap b)\ c_1\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ (P \cap -\ b)\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ (Cond\ b\ c_1\ c_2)\ Q,A$

| *While*: $[\![wf\ r;\ \forall \sigma.\ \Gamma,\Theta\vdash_{t/F}\ (\{\sigma\} \cap P \cap b)\ c\ (\{t.\ (t,\sigma)\in r\} \cap P),A]\!]$
$\Longrightarrow$

$\Gamma,\Theta\vdash_{t/F}$ P (*While* b c) ($P \cap -$ b),A

| *Guard*: $\Gamma,\Theta\vdash_{t/F}$ ($g \cap P$) c Q,A
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ ($g \cap P$) *Guard* f g c Q,A

| *Guarantee*: $[\![f \in F; \Gamma,\Theta\vdash_{t/F} (g \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ P (*Guard* f g c) Q,A

| *CallRec*:
$[\![(P,p,Q,A) \in$ *Specs*;
wf r;
Specs-wf = ($\lambda p\ \sigma$. ($\lambda(P,q,Q,A)$. ($P \cap \{s.\ ((s,q),(\sigma,p)) \in r\}$,$q$,$Q$,$A$)) ‘ *Specs*);
$\forall (P,p,Q,A)\in$ *Specs*.
$p \in$ *dom* $\Gamma \wedge (\forall \sigma.\ \Gamma,\Theta \cup$ *Specs-wf* p $\sigma\vdash_{t/F}$ ($\{\sigma\} \cap P$) (*the* (Γ p)) Q,A)
$]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ P (*Call* p) Q,A

| *DynCom*: $\forall s \in P$. $\Gamma,\Theta\vdash_{t/F}$ P (c s) Q,A
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ P (*DynCom* c) Q,A

| *Throw*: $\Gamma,\Theta\vdash_{t/F}$ A *Throw* Q,A

| *Catch*: $[\![\Gamma,\Theta\vdash_{t/F} P\ c_1\ Q,R; \Gamma,\Theta\vdash_{t/F} R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}$ P *Catch* c_1 c_2 Q,A

| *Conseq*: $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{t/F} P'\ c\ Q',A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}$ P c Q,A

| *Asm*: (P,p,Q,A) $\in \Theta$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ P (*Call* p) Q,A

| *ExFalso*: $[\![\Gamma,\Theta\models_{t/F} P\ c\ Q,A; \neg\ \Gamma\models_{t/F} P\ c\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}$ P c Q,A
— This is a hack rule that enables us to derive completeness for an arbitrary context Θ, from completeness for an empty context.

Does not work, because of rule ExFalso, the context Θ is to blame. A weaker version with empty context can be derived from soundness later on.

lemma *hoaret-to-hoarep*:
assumes *hoaret*: $\Gamma,\Theta\vdash_{t/F}$ P p Q,A
shows $\Gamma,\Theta\vdash_{/F}$ P p Q,A

using *hoaret*
proof (*induct*)
  case *Skip* **thus** *?case* **by** (*rule hoarep.intros*)
next
  case *Basic* **thus** *?case* **by** (*rule hoarep.intros*)
next
  case *Seq* **thus** *?case* **by** − (*rule hoarep.intros*)
next
  case *Cond* **thus** *?case* **by** − (*rule hoarep.intros*)
next
  case (*While r* Θ *F P b c A*)
  hence $\forall\sigma.\ \Gamma,\Theta\vdash_{/F}$ $(\{\sigma\} \cap P \cap b)\ c\ (\{t.\ (t, \sigma) \in r\} \cap P),A$
    by *iprover*
  hence $\Gamma,\Theta\vdash_{/F}$ $(P \cap b)\ c\ P,A$
    by (*rule HoarePartialDef.conseq*) *blast*
  then show $\Gamma,\Theta\vdash_{/F}$ P *While* $b\ c\ (P \cap -\ b),A$
    by (*rule hoarep.While*)
next
  case *Guard* **thus** *?case* **by** − (*rule hoarep.intros*)

next
  case *DynCom* **thus** *?case* **by** (*blast intro*: *hoarep.DynCom*)
next
  case *Throw* **thus** *?case* **by** − (*rule hoarep.Throw*)
next
  case *Catch* **thus** *?case* **by** − (*rule hoarep.Catch*)
next
  case *Conseq* **thus** *?case* **by** − (*rule hoarep.Conseq*,*blast*)
next
  case *Asm* **thus** *?case* **by** (*rule HoarePartialDef.Asm*)
next
  case (*ExFalso* Θ *F P c Q A*)
  assume $\Gamma,\Theta\models_{t/F}$ $P\ c\ Q,A$
  hence $\Gamma,\Theta\models_{/F}$ $P\ c\ Q,A$
    oops

lemma *hoaret-augment-context*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ $P\ p\ Q,A$
  shows $\bigwedge\Theta'.\ \Theta \subseteq \Theta' \Longrightarrow \Gamma,\Theta'\vdash_{t/F}$ $P\ p\ Q,A$
using *deriv*
proof (*induct*)
  case (*CallRec P p Q A Specs r Specs-wf* Θ *F* Θ′)
  have *aug*: $\Theta \subseteq \Theta'$ **by** *fact*
  then
  have *h*: $\bigwedge\tau\ p.\ \Theta \cup$ *Specs-wf* $p\ \tau$
      $\subseteq \Theta' \cup$ *Specs-wf* $p\ \tau$
    by *blast*

have $\forall (P,p,Q,A)\in Specs.\ p \in dom\ \Gamma \wedge$
$(\forall \tau.\ \Gamma,\Theta \cup Specs\text{-}wf\ p\ \tau \vdash_{t/F} (\{\tau\} \cap P)\ (the\ (\Gamma\ p))\ Q,A \wedge$
$(\forall x.\ \Theta \cup Specs\text{-}wf\ p\ \tau$
$\subseteq x \longrightarrow$
$\Gamma,x\vdash_{t/F} (\{\tau\} \cap P)\ (the\ (\Gamma\ p))\ Q,A))$ **by** *fact*
hence $\forall (P,p,Q,A)\in Specs.\ p \in dom\ \Gamma \wedge$
$(\forall \tau.\ \Gamma,\Theta' \cup Specs\text{-}wf\ p\ \tau \vdash_{t/F} (\{\tau\} \cap P)\ (the\ (\Gamma\ p))\ Q,A)$
apply (*clarify*)
apply (*rename-tac P p Q A*)
apply (*drule* (*1*) *bspec*)
apply (*clarsimp*)
apply (*erule-tac* $x{=}\tau$ **in** *allE*)
apply *clarify*
apply (*erule-tac* $x{=}\Theta' \cup Specs\text{-}wf\ p\ \tau$ **in** *allE*)
apply (*insert aug*)
apply *auto*
done
with *CallRec* **show** *?case* **by** − (*rule hoaret.CallRec*)
next
case *DynCom* **thus** *?case* **by** (*blast intro*: *hoaret.DynCom*)
next
case (*Conseq* $P\ \Theta\ F\ c\ Q\ A\ \Theta'$)
from *Conseq*
have $\forall s \in P.\ (\exists P'\ Q'\ A'.\ (\Gamma,\Theta' \vdash_{t/F} P'\ c\ Q',A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
with *Conseq* **show** *?case* **by** − (*rule hoaret.Conseq*)
next
case (*ExFalso* $\Theta\ F\ P\ \ c\ Q\ A\ \Theta'$)
have $\Gamma,\Theta\models_{t/F} P\ c\ Q,A\ \neg\ \Gamma\models_{t/F} P\ c\ Q,A\ \Theta \subseteq \Theta'$ **by** *fact+*
then show *?case*
by (*fastforce intro*: *hoaret.ExFalso simp add*: *cvalidt-def*)
qed (*blast intro*: *hoaret.intros*)+

14.4 Some Derived Rules

lemma *Conseq'*: $\forall s.\ s \in P \longrightarrow$
$(\exists P'\ Q'\ A'.$
$(\forall\ Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)) \wedge$
$(\exists Z.\ s \in P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A)))$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
apply (*rule Conseq*)
apply (*rule ballI*)
apply (*erule-tac* $x{=}s$ **in** *allE*)
apply (*clarify*)
apply (*rule-tac* $x{=}P'\ Z$ **in** *exI*)
apply (*rule-tac* $x{=}Q'\ Z$ **in** *exI*)

apply (*rule-tac* $x=A'$ Z **in** *exI*)
apply *blast*
done

lemma *conseq*:$[\![\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\forall s.\ s \in P \longrightarrow (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\Longrightarrow$
$\Gamma,\Theta{\vdash}_{t/F}\ P\ c\ Q,A$
by (*rule Conseq*) *blast*

theorem *conseqPrePost*:
$\Gamma,\Theta{\vdash}_{t/F}\ P'\ c\ Q',A' \Longrightarrow P \subseteq P' \Longrightarrow\ Q' \subseteq Q \Longrightarrow A' \subseteq A \Longrightarrow\ \Gamma,\Theta{\vdash}_{t/F}\ P\ c$
Q,A
by (*rule conseq* [**where** $?P'{=}\lambda Z.\ P'$ **and** $?Q'{=}\lambda Z.\ Q'$]) *auto*

lemma *conseqPre*: $\Gamma,\Theta{\vdash}_{t/F}\ P'\ c\ Q,A \Longrightarrow P \subseteq P' \Longrightarrow \Gamma,\Theta{\vdash}_{t/F}\ P\ c\ Q,A$
by (*rule conseq*) *auto*

lemma *conseqPost*: $\Gamma,\Theta{\vdash}_{t/F}\ P\ c\ Q',A' \Longrightarrow Q' \subseteq Q \Longrightarrow A' \subseteq A \Longrightarrow\ \Gamma,\Theta{\vdash}_{t/F}\ P$
$c\ Q,A$
by (*rule conseq*) *auto*

lemma *Spec-wf-conv*:
$(\lambda(P,\ q,\ Q,\ A).\ (P \cap \{s.\ ((s,\ q),\ \tau,\ p) \in r\},\ q,\ Q,\ A))$ ‘
$(\bigcup p{\in}Procs.\ \bigcup Z.\ \{(P\ p\ Z,\ p,\ Q\ p\ Z,\ A\ p\ Z)\}) =$
$(\bigcup q{\in}Procs.\ \bigcup Z.\ \{(P\ q\ Z \cap \{s.\ ((s,\ q),\ \tau,\ p) \in r\},\ q,\ Q\ q\ Z,\ A\ q\ Z)\})$
by (*auto intro*!: *image-eqI*)

lemma *CallRec'*:
$[\![p{\in}Procs;\ Procs \subseteq dom\ \Gamma;$
$wf\ r;$
$\forall p{\in}Procs.\ \forall \tau\ Z.$
$\Gamma,\Theta{\cup}(\bigcup q{\in}Procs.\ \bigcup Z.$
$\{((P\ q\ Z) \cap \{s.\ ((s,q),(\tau,p)) \in r\},q,Q\ q\ Z,(A\ q\ Z))\})$
$\vdash_{t/F} (\{\tau\} \cap (P\ p\ Z))\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)]\!]$
$\Longrightarrow$
$\Gamma,\Theta{\vdash}_{t/F}\ (P\ p\ Z)\ (Call\ p)\ (Q\ p\ Z),(A\ p\ Z)$
apply (*rule CallRec* [**where** $Specs{=}\bigcup p{\in}Procs.\ \bigcup Z.\ \{((P\ p\ Z),p,Q\ p\ Z,A\ p\ Z)\}$
and
$r{=}r$])
apply *blast*
apply *assumption*
apply (*rule refl*)
apply (*clarsimp*)
apply (*rename-tac* p')
apply (*rule conjI*)
apply *blast*

apply (*intro allI*)
apply (*rename-tac Z* τ)
apply (*drule-tac* $x{=}p'$ **in** *bspec*, *assumption*)
apply (*erule-tac* $x{=}\tau$ **in** *allE*)
apply (*erule-tac* $x{=}Z$ **in** *allE*)
apply (*fastforce simp add*: *Spec-wf-conv*)
done

end

15 Properties of Total Correctness Hoare Logic

theory *HoareTotalProps* **imports** *SmallStep HoareTotalDef HoarePartialProps* **begin**

15.1 Soundness

lemma *hoaret-sound*:
 assumes *hoare*: $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
 shows $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
using *hoare*
proof (*induct*)
  case (*Skip* Θ *F P A*)
  show $\Gamma,\Theta \models_{t/F} P\ \mathit{Skip}\ P,A$
  proof (*rule cvalidtI*)
    fix *s t*
    assume $\Gamma \vdash \langle \mathit{Skip},\mathit{Normal}\ s\rangle \Rightarrow t\ s \in P$
    thus $t \in \mathit{Normal}\ `\ P \cup \mathit{Abrupt}\ `\ A$
      by *cases auto*
  next
    fix *s* **show** $\Gamma \vdash \mathit{Skip} \downarrow \mathit{Normal}\ s$
      by (*rule terminates.intros*)
  qed
next
  case (*Basic* Θ *F f P A*)
  show $\Gamma,\Theta \models_{t/F} \{s.\ f\ s \in P\}\ (\mathit{Basic}\ f)\ P,A$
  proof (*rule cvalidtI*)
    fix *s t*
    assume $\Gamma \vdash \langle \mathit{Basic}\ f,\mathit{Normal}\ s\rangle \Rightarrow t\ s \in \{s.\ f\ s \in P\}$
    thus $t \in \mathit{Normal}\ `\ P \cup \mathit{Abrupt}\ `\ A$
      by *cases auto*
  next
    fix *s* **show** $\Gamma \vdash \mathit{Basic}\ f \downarrow \mathit{Normal}\ s$
      by (*rule terminates.intros*)
  qed
next
  case (*Spec* Θ *F r Q A*)
  show $\Gamma,\Theta \models_{t/F} \{s.\ (\forall t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,\ t) \in r)\}\ \mathit{Spec}\ r\ Q,A$

proof (*rule cvalidtI*)
  fix *s t*
  assume $\Gamma\vdash\langle Spec\ r\ ,Normal\ s\rangle \Rightarrow t$
    $s \in \{s.\ (\forall t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,\ t) \in r)\}$
  thus $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
    by *cases auto*
next
  fix *s* **show** $\Gamma\vdash Spec\ r \downarrow Normal\ s$
    by (*rule terminates.intros*)
qed
next
case (*Seq* Θ *F P c1 R A c2 Q*)
have *valid-c1*: $\Gamma,\Theta \models_{t/F} P\ c1\ R,A$ **by** *fact*
have *valid-c2*: $\Gamma,\Theta \models_{t/F} R\ c2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models_{t/F} P\ Seq\ c1\ c2\ Q,A$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *exec P* **obtain** *r* **where**
    exec-c1: $\Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow r$ **and** *exec-c2*: $\Gamma\vdash\langle c2,r\rangle \Rightarrow t$
    by *cases auto*
  with *t-notin-F* **have** $r \notin Fault\ `\ F$
    by (*auto dest*: *Fault-end*)
  from *valid-c1 ctxt exec-c1 P this*
  have *r*: $r \in Normal\ `\ R \cup Abrupt\ `\ A$
    by (*rule cvalidt-postD*)
  show $t\in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases r*)
    case (*Normal* r')
    with *exec-c2 r*
    show $t\in Normal\ `\ Q \cup Abrupt\ `\ A$
      apply –
      apply (*rule cvalidt-postD* [*OF valid-c2 ctxt - - t-notin-F*])
      apply *auto*
      done
  next
    case (*Abrupt* r')
    with *exec-c2* **have** $t=Abrupt\ r'$
      by (*auto elim*: *exec-elim-cases*)
    with *Abrupt r* **show** *?thesis*
      by *auto*
  next
    case *Fault* **with** *r* **show** *?thesis* **by** *blast*
  next
    case *Stuck* **with** *r* **show** *?thesis* **by** *blast*
  qed

next
fix s
assume $ctxt$: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
assume P: $s\in P$
show $\Gamma\vdash Seq\ c1\ c2 \downarrow Normal\ s$
proof −
from *valid-c1 ctxt P*
have $\Gamma\vdash c1\downarrow Normal\ s$
by (*rule cvalidt-termD*)
moreover
{
fix r **assume** *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow r$
have $\Gamma\vdash c2 \downarrow r$
proof (*cases r*)
case (*Normal r′*)
with *cvalidt-postD* [*OF valid-c1 ctxt exec-c1 P*]
have r: $r\in Normal$ ‘ R
by *auto*
with *cvalidt-termD* [*OF valid-c2 ctxt*] *exec-c1*
show $\Gamma\vdash c2 \downarrow r$
by *auto*
qed *auto*
}
ultimately show *?thesis*
by (*iprover intro*: *terminates.intros*)
qed
qed
next
case (*Cond* Θ *F P b c1 Q A c2*)
have *valid-c1*: $\Gamma,\Theta \models_{t/F} (P \cap b)\ c1\ Q,A$ **by** *fact*
have *valid-c2*: $\Gamma,\Theta \models_{t/F} (P \cap -\ b)\ c2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models_{t/F} P\ Cond\ b\ c1\ c2\ Q,A$
proof (*rule cvalidtI*)
fix $s\ t$
assume $ctxt$: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
assume $exec$: $\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow t$
assume P: $s \in P$
assume *t-notin-F*: $t \notin Fault$ ‘ F
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
proof (*cases* $s\in b$)
case *True*
with *exec* **have** $\Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow t$
by *cases auto*
with *P True*
show *?thesis*
by − (*rule cvalidt-postD* [*OF valid-c1 ctxt - - t-notin-F*],*auto*)
next
case *False*

    with *exec P* **have** $\Gamma\vdash\langle c2, Normal\ s\rangle \Rightarrow t$
      by *cases auto*
    with *P False*
    show *?thesis*
      by $-$ (*rule cvalidt-postD* [*OF valid-c2 ctxt - - t-notin-F*],*auto*)
  qed
 next
  fix *s*
  assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *P*: $s \in P$
  thus $\Gamma\vdash Cond\ b\ c1\ c2 \downarrow Normal\ s$
    using *cvalidt-termD* [*OF valid-c1 ctxt*] *cvalidt-termD* [*OF valid-c2 ctxt*]
    by (*cases* $s \in b$) (*auto intro*: *terminates.intros*)
 qed
next
 case (*While* $r\ \Theta\ F\ P\ b\ c\ A$)
 assume *wf*: *wf r*
 have *valid-c*: $\forall\sigma.\ \Gamma,\Theta\models_{t/F} (\{\sigma\} \cap P \cap b)\ c\ (\{t.\ (t, \sigma) \in r\} \cap P),A$
  using *While.hyps* **by** *iprover*
 show $\Gamma,\Theta \models_{t/F} P\ (While\ b\ c)\ (P \cap -\ b),A$
 proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
  assume *wprems*: $\Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow t\ s \in P\ t \notin Fault\ `\ F$
  from *wf*
  have $\bigwedge t.\ [\![\Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow t;\ s \in P;\ t \notin Fault\ `\ F]\!]$
       $\Longrightarrow t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$
  proof (*induct*)
   fix *s t*
   assume *hyp*:
    $\bigwedge s'\ t.\ [\![(s',s)\in r;\ \Gamma\vdash\langle While\ b\ c, Normal\ s'\rangle \Rightarrow t;\ s' \in P;\ t \notin Fault\ `\ F]\!]$
       $\Longrightarrow t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$
   assume *exec*: $\Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow t$
   assume *P*: $s \in P$
   assume *t-notin-F*: $t \notin Fault\ `\ F$
   from *exec*
   show $t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$
   proof (*cases*)
    fix s'
    assume *b*: $s\in b$
    assume *exec-c*: $\Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow s'$
    assume *exec-w*: $\Gamma\vdash\langle While\ b\ c, s'\rangle \Rightarrow t$
    from *exec-w t-notin-F* **have** $s' \notin Fault\ `\ F$
     by (*auto dest*: *Fault-end*)
    from *exec-c P b valid-c ctxt this*
    have s': $s' \in Normal\ `\ (\{s'.\ (s', s) \in r\} \cap P) \cup Abrupt\ `\ A$
     by (*auto simp add*: *cvalidt-def validt-def valid-def*)
    show *?thesis*
    proof (*cases* s')

case *Normal*
with *exec-w* s' *t-notin-F*
show *?thesis*
by − (*rule hyp*,*auto*)
next
case *Abrupt*
with *exec-w* **have** $t{=}s'$
by (*auto dest*: *Abrupt-end*)
with *Abrupt* s' **show** *?thesis*
by *blast*
next
case *Fault*
with *exec-w* **have** $t{=}s'$
by (*auto dest*: *Fault-end*)
with *Fault* s' **show** *?thesis*
by *blast*
next
case *Stuck*
with *exec-w* **have** $t{=}s'$
by (*auto dest*: *Stuck-end*)
with *Stuck* s' **show** *?thesis*
by *blast*
qed
next
assume $s{\notin}b$ $t{=}Normal\ s$ **with** P **show** *?thesis* **by** *simp*
qed
qed
with *wprems* **show** $t \in Normal\ `\ (P \cap -\ b) \cup Abrupt\ `\ A$ **by** *blast*
next
fix s
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}_{t/F}\ P\ (Call\ p)\ Q{,}A$
assume $s \in P$
with *wf*
show $\Gamma{\vdash}While\ b\ c \downarrow Normal\ s$
proof (*induct*)
fix s
assume *hyp*: $\bigwedge s'.\ [\![(s',s){\in}r;\ s' \in P]\!]$
$\Longrightarrow \Gamma{\vdash}While\ b\ c \downarrow Normal\ s'$
assume P: $s \in P$
show $\Gamma{\vdash}While\ b\ c \downarrow Normal\ s$
proof (*cases* $s \in b$)
case *False* **with** P **show** *?thesis*
by (*blast intro*: *terminates.intros*)
next
case *True*
with *valid-c P ctxt*
have $\Gamma{\vdash}c \downarrow Normal\ s$
by (*simp add*: *cvalidt-def validt-def*)
moreover

{
  fix s'
  assume *exec-c*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow s'$
  have $\Gamma\vdash$ *While b c* $\downarrow s'$
  proof (*cases* s')
    case (*Normal* s'')
    with *exec-c P True valid-c ctxt*
    have s': $s' \in Normal$ ' $(\{s'.\ (s', s) \in r\} \cap P)$
      by (*fastforce simp add*: *cvalidt-def validt-def valid-def*)
    then show *?thesis*
      by (*blast intro*: *hyp*)
  qed *auto*
}
ultimately
show *?thesis*
  by (*blast intro*: *terminates.intros*)
qed
qed
qed
next
case (*Guard* Θ *F g P c Q A f*)
have *valid-c*: $\Gamma,\Theta \models_{t/F} (g \cap P)\ c\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models_{t/F} (g \cap P)$ *Guard f g c Q*,*A*
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow t$
  assume *t-notin-F*: $t \notin Fault$ ' F
  assume P:$s \in (g \cap P)$
  from *exec P* **have** $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
    by *cases auto*
  from *valid-c ctxt this P t-notin-F*
  show $t \in Normal$ ' $Q \cup Abrupt$ ' A
    by (*rule cvalidt-postD*)
next
  fix *s*
  assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume P:$s \in (g \cap P)$
  thus $\Gamma\vdash$*Guard f g c* $\downarrow$ *Normal s*
    by (*auto intro*: *terminates.intros cvalidt-termD* [*OF valid-c ctxt*])
qed
next
case (*Guarantee f F* Θ *g P c Q A*)
have *valid-c*: $\Gamma,\Theta \models_{t/F} (g \cap P)\ c\ Q,A$ **by** *fact*
have *f-F*: $f \in F$ **by** *fact*
show $\Gamma,\Theta \models_{t/F} P$ *Guard f g c Q*,*A*
proof (*rule cvalidtI*)
  fix *s t*

assume *ctxt*: $\forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q{,}A$
assume *exec*: $\Gamma\vdash\langle \mathit{Guard}\ f\ g\ c{,}\mathit{Normal}\ s\rangle \Rightarrow t$
assume *t-notin-F*: $t \notin \mathit{Fault}$ ' F
assume P:$s \in P$
from *exec f-F t-notin-F* **have** g: $s \in g$
by *cases auto*
with P **have** P': $s \in g \cap P$
by *blast*
from *exec g* **have** $\Gamma\vdash\langle c{,}\mathit{Normal}\ s\rangle \Rightarrow t$
by *cases auto*
from *valid-c ctxt this* P' *t-notin-F*
show $t \in \mathit{Normal}$ ' $Q \cup \mathit{Abrupt}$ ' A
by (*rule cvalidt-postD*)
next
fix s
assume *ctxt*: $\forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q{,}A$
assume P:$s \in P$
thus $\Gamma\vdash \mathit{Guard}\ f\ g\ c\ \downarrow \mathit{Normal}\ s$
by (*auto intro*: *terminates.intros cvalidt-termD* [*OF valid-c ctxt*])
qed
next
case (*CallRec P p Q A Specs r Specs-wf* Θ *F*)
have p: $(P{,}p{,}Q{,}A) \in \mathit{Specs}$ **by** *fact*
have *wf*: *wf r* **by** *fact*
have *Specs-wf*:
$\mathit{Specs\text{-}wf} = (\lambda p\ \tau.\ (\lambda(P{,}q{,}Q{,}A).\ (P \cap \{s.\ ((s,\ q){,}\tau{,}p) \in r\}{,}q{,}Q{,}A))$ ' $\mathit{Specs})$ **by** *fact*
from *CallRec.hyps*
have *valid-body*:
$\forall(P,\ p,\ Q,\ A)\in\mathit{Specs}.\ p \in \mathit{dom}\ \Gamma \wedge$
$(\forall\tau.\ \Gamma{,}\Theta \cup \mathit{Specs\text{-}wf}\ p\ \tau\models_{t/F} (\{\tau\} \cap P)\ \mathit{the}\ (\Gamma\ p)\ Q{,}A)$ **by** *auto*
show $\Gamma{,}\Theta \models_{t/F} P\ (\mathit{Call}\ p)\ Q{,}A$
proof −
{
fix τp
assume *ctxt*: $\forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F} P\ (\mathit{Call}\ p)\ Q{,}A$
from *wf*
have $\bigwedge\tau\ p\ P\ Q\ A.\ [\![\tau p = (\tau{,}p);\ (P{,}p{,}Q{,}A) \in \mathit{Specs}]\!] \Longrightarrow$
$\Gamma\models_{t/F} (\{\tau\} \cap P)\ (\mathit{the}\ (\Gamma\ (p)))\ Q{,}A$
proof (*induct* τp *rule*: *wf-induct* [*rule-format*, *consumes 1*, *case-names WF*])
case (*WF* $\tau p\ \tau\ p\ P\ Q\ A$)
have τp: $\tau p = (\tau,\ p)$ **by** *fact*
have p: $(P,\ p,\ Q,\ A) \in \mathit{Specs}$ **by** *fact*
{
fix $q\ P'\ Q'\ A'$
assume q: $(P'{,}q{,}Q'{,}A') \in \mathit{Specs}$
have $\Gamma\models_{t/F} (P' \cap \{s.\ ((s{,}q),\ \tau{,}p) \in r\})\ (\mathit{Call}\ q)\ Q'{,}A'$
proof (*rule validtI*)

fix *s t*
assume *exec-q*:
$\Gamma\vdash\langle Call\ q, Normal\ s\rangle \Rightarrow t$
assume *Pre*: $s \in P' \cap \{s.\ ((s,q),\ \tau,p) \in r\}$
assume *t-notin-F*: $t \notin Fault\ `\ F$
from *Pre q* τp
have *valid-bdy*:
$\Gamma\models_{t/F} (\{s\} \cap P')\ the\ (\Gamma\ q)\ Q',A'$
by $-$ (*rule WF.hyps*, *auto*)
from *Pre q*
have *Pre'*: $s \in \{s\} \cap P'$
by *auto*
from *exec-q* **show** $t \in Normal\ `\ Q' \cup Abrupt\ `\ A'$
proof (*cases*)
fix *bdy*
assume *bdy*: $\Gamma\ q = Some\ bdy$
assume *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ s\rangle \Rightarrow t$
from *valid-bdy* [*simplified bdy option.sel*] *t-notin-F exec-bdy Pre'*
have $t \in Normal\ `\ Q' \cup Abrupt\ `\ A'$
by (*auto simp add*: *validt-def valid-def*)
with *Pre q*
show *?thesis*
by *auto*
next
assume $\Gamma\ q = None$
with *q valid-body* **have** *False* **by** *auto*
thus *?thesis* **..**
qed
next
fix *s*
assume *Pre*: $s \in P' \cap \{s.\ ((s,q),\ \tau,p) \in r\}$
from *Pre q* τp
have *valid-bdy*:
$\Gamma \models_{t/F} (\{s\} \cap P')\ (the\ (\Gamma\ q))\ Q',A'$
by $-$ (*rule WF.hyps*, *auto*)
from *Pre q*
have *Pre'*: $s \in \{s\} \cap P'$
by *auto*
from *valid-bdy ctxt Pre'*
have $\Gamma\vdash the\ (\Gamma\ q) \downarrow Normal\ s$
by (*auto simp add*: *validt-def*)
with *valid-body q*
show $\Gamma\vdash Call\ q\downarrow\ Normal\ s$
by (*fastforce intro*: *terminates.Call*)
qed
}
hence $\forall (P,\ p,\ Q,\ A){\in}Specs\text{-}wf\ p\ \tau.\ \Gamma\models_{t/F} P\ Call\ p\ Q,A$
by (*auto simp add*: *cvalidt-def Specs-wf*)
with *ctxt* **have** $\forall (P,\ p,\ Q,\ A){\in}\Theta \cup Specs\text{-}wf\ p\ \tau.\ \Gamma\models_{t/F} P\ Call\ p\ Q,A$

by *auto*
with *p valid-body*
show $\Gamma \models_{t/F} (\{\tau\} \cap P)$ (*the* (Γ *p*)) Q,A
by (*simp add*: *cvalidt-def*) *blast*
qed
}
note *lem* = *this*
have *valid-body'*:
$\bigwedge\tau. \forall (P, p, Q, A)\in\Theta. \Gamma\models_{t/F} P$ (*Call p*) $Q,A \Longrightarrow$
$\forall (P,p,Q,A)\in$*Specs*. $\Gamma\models_{t/F} (\{\tau\} \cap P)$ (*the* (Γ *p*)) Q,A
by (*auto intro*: *lem*)
show $\Gamma,\Theta \models_{t/F} P$ (*Call p*) Q,A
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta. \Gamma\models_{t/F} P$ (*Call p*) Q,A
assume *exec-call*: $\Gamma\vdash\langle$*Call p*,*Normal s*$\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ F
from *exec-call* **show** $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
proof (*cases*)
fix *bdy*
assume *bdy*: Γ *p* = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal s*$\rangle \Rightarrow t$
from *exec-body bdy p P t-notin-F*
valid-body' [*of s*, *OF ctxt*]
ctxt
have $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
apply (*simp only*: *cvalidt-def validt-def valid-def*)
apply (*drule* (*1*) *bspec*)
apply *auto*
done
with *p P*
show *?thesis*
by *simp*
next
assume Γ *p* = *None*
with *p valid-body* **have** *False* **by** *auto*
thus *?thesis* **by** *simp*
qed
next
fix *s*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta. \Gamma\models_{t/F} P$ (*Call p*) Q,A
assume *P*: $s \in P$
show $\Gamma\vdash$*Call p* $\downarrow$ *Normal s*
proof −
from *ctxt P p valid-body'* [*of s*,*OF ctxt*]
have $\Gamma\vdash$(*the* (Γ *p*)) $\downarrow$ *Normal s*
by (*auto simp add*: *cvalidt-def validt-def*)

```
      with valid-body p show ?thesis
        by (fastforce intro: terminates.Call)
    qed
  qed
 qed
next
 case (DynCom P Θ F c Q A)
 hence valid-c: ∀ s∈P. Γ,Θ⊨t/F P (c s) Q,A by simp
 show Γ,Θ⊨t/F P DynCom c Q,A
 proof (rule cvalidtI)
   fix s t
   assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ⊨t/F P (Call p) Q,A
   assume exec: Γ⊢⟨DynCom c,Normal s⟩ ⇒ t
   assume P: s ∈ P
   assume t-notin-F: t ∉ Fault ‘ F
   from exec show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
   proof (cases)
     assume Γ⊢⟨c s,Normal s⟩ ⇒ t
     from cvalidt-postD [OF valid-c [rule-format, OF P] ctxt this P t-notin-F]
     show ?thesis .
   qed
 next
   fix s
   assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ⊨t/F P (Call p) Q,A
   assume P: s ∈ P
   show Γ⊢DynCom c ↓ Normal s
   proof −
     from cvalidt-termD [OF valid-c [rule-format, OF P] ctxt P]
     have Γ⊢c s ↓ Normal s .
     thus ?thesis
       by (rule terminates.intros)
   qed
 qed
next
 case (Throw Θ F A Q)
 show Γ,Θ ⊨t/F A Throw Q,A
 proof (rule cvalidtI)
   fix s t
   assume Γ⊢⟨Throw,Normal s⟩ ⇒ t s ∈ A
   then show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
     by cases simp
 next
   fix s
   show Γ⊢Throw ↓ Normal s
     by (rule terminates.intros)
 qed
next
 case (Catch Θ F P c1 Q R c2 A)
```

have *valid-c1*: $\Gamma,\Theta \models_{t/F} P\ c_1\ Q,R$ **by** *fact*
have *valid-c2*: $\Gamma,\Theta \models_{t/F} R\ c_2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models_{t/F} P$ *Catch* $c_1\ c_2\ Q,A$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle Catch\ c_1\ c_2, Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *exec* **show** $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases*)
    fix s'
    assume *exec-c1*: $\Gamma\vdash\langle c_1, Normal\ s\rangle \Rightarrow Abrupt\ s'$
    assume *exec-c2*: $\Gamma\vdash\langle c_2, Normal\ s'\rangle \Rightarrow t$
    from *cvalidt-postD* [*OF valid-c1 ctxt exec-c1 P*]
    have $Abrupt\ s' \in Abrupt\ `\ R$
      by *auto*
    with *cvalidt-postD* [*OF valid-c2 ctxt*] *exec-c2 t-notin-F*
    show *?thesis*
      by *fastforce*
  next
    assume *exec-c1*: $\Gamma\vdash\langle c_1, Normal\ s\rangle \Rightarrow t$
    assume *notAbr*: $\neg$ *isAbr t*
    from *cvalidt-postD* [*OF valid-c1 ctxt exec-c1 P*] *t-notin-F*
    have $t \in Normal\ `\ Q \cup Abrupt\ `\ R$ **.**
    with *notAbr*
    show *?thesis*
      by *auto*
  qed
next
  fix *s*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *P*: $s \in P$
  show $\Gamma\vdash Catch\ c_1\ c_2 \downarrow Normal\ s$
  proof −
    from *valid-c1 ctxt P*
    have $\Gamma\vdash c_1\downarrow Normal\ s$
      by (*rule cvalidt-termD*)
    moreover
    {
      fix *r* **assume** *exec-c1*: $\Gamma\vdash\langle c_1, Normal\ s\rangle \Rightarrow Abrupt\ r$
      from *cvalidt-postD* [*OF valid-c1 ctxt exec-c1 P*]
      have *r*: $Abrupt\ r{\in}Normal\ `\ Q \cup Abrupt\ `\ R$
        by *auto*
      hence $Abrupt\ r{\in}Abrupt\ `\ R$ **by** *fast*
      with *cvalidt-termD* [*OF valid-c2 ctxt*] *exec-c1*
      have $\Gamma\vdash c_2 \downarrow Normal\ r$
        by *fast*

}
ultimately show *?thesis*
  by (*iprover intro*: *terminates.intros*)
qed
qed
next
case (*Conseq* P Θ F c Q A)
hence *adapt*:
$\forall s \in P.\ (\exists P'\ Q'\ A'.\ (\Gamma,\Theta \models_{t/F} P'\ c\ Q',A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
show $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
proof (*rule cvalidtI*)
fix s t
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault$ ‘ F
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
proof −
from *adapt* [*rule-format*, *OF P*]
obtain P' **and** Q' **and** A' **where**
valid-P'-Q': $\Gamma,\Theta \models_{t/F} P'\ c\ Q',A'$
and *weaken*: $s \in P'$ $Q' \subseteq$ Q $A' \subseteq A$
by *blast*
from *exec valid-P'-Q' ctxt t-notin-F*
have *P'-Q'*: $Normal\ s \in Normal$ ‘ $P' \longrightarrow$
$t \in Normal$ ‘ $Q' \cup Abrupt$ ‘ A'
by (*unfold cvalidt-def validt-def valid-def*) *blast*
hence $s \in P' \longrightarrow t \in Normal$ ‘ $Q' \cup Abrupt$ ‘ A'
by *blast*
with *weaken*
show *?thesis*
by *blast*
qed
next
fix s
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
assume *P*: $s \in P$
show $\Gamma\vdash c \downarrow Normal\ s$
proof −
from *P adapt*
obtain P' **and** Q' **and** A' **where**
$\Gamma,\Theta \models_{t/F} P'\ c\ Q',A'$
$s \in P'$
by *blast*
with *ctxt*
show *?thesis*
by (*simp add*: *cvalidt-def validt-def*)

qed
qed
next
case $(Asm\ P\ p\ Q\ A\ \Theta\ F)$
assume $(P,\ p,\ Q,\ A) \in \Theta$
then show $\Gamma,\Theta \models_{t/F} P\ (Call\ p)\ Q,A$
by (*auto simp add*: *cvalidt-def*)
next
case *ExFalso* **thus** *?case* **by** *iprover*
qed

lemma *hoaret-sound'*:
$\Gamma,\{\}\vdash_{t/F} P\ c\ Q,A \Longrightarrow \Gamma\models_{t/F} P\ c\ Q,A$
apply (*drule hoaret-sound*)
apply (*simp add*: *cvalidt-def*)
done

theorem *total-to-partial*:
assumes *total*: $\Gamma,\{\}\vdash_{t/F} P\ c\ Q,A$ **shows** $\Gamma,\{\}\vdash_{/F} P\ c\ Q,A$
proof −
from *total* **have** $\Gamma,\{\}\models_{t/F} P\ c\ Q,A$
by (*rule hoaret-sound*)
hence $\Gamma\models_{/F} P\ c\ Q,A$
by (*simp add*: *cvalidt-def validt-def cvalid-def*)
thus *?thesis*
by (*rule hoare-complete*)
qed

15.2 Completeness

lemma *MGT-valid*:
$\Gamma\models_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$
$s\}\ c$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule validtI*)
fix $s\ t$
assume $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
$s \in \{s.\ s = Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$
$s\}$
$t \notin Fault\ `\ F$
thus $t \in Normal\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\} \cup$
$Abrupt\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*cases t*)
apply (*auto simp add*: *final-notin-def*)
done
next
fix s
assume $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$

$s\}$
  thus $\Gamma\vdash c\downarrow Normal\ s$
    by *blast*
qed

The consequence rule where the existential Z is instantiated to s. Usefull in proof of *MGT-lemma*.

lemma *ConseqMGT*:
  assumes *modif*: $\forall Z::'a.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z::'a\ assn)\ c\ (Q'\ Z),(A'\ Z)$
  assumes *impl*: $\bigwedge s.\ s \in P \Longrightarrow s \in P'\ s \wedge (\forall t.\ t \in Q'\ s \longrightarrow t \in Q) \wedge$
$(\forall t.\ t \in A'\ s \longrightarrow t \in A)$
  shows $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
using *impl*
by $-$ (*rule conseq* [*OF modif*],*blast*)

lemma *MGT-implies-complete*:
  assumes *MGT*: $\forall Z.\ \Gamma,\{\}\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault$
$`\ (-F)) \wedge$
$\Gamma\vdash c\downarrow Normal\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  assumes *valid*: $\Gamma \models_{t/F} P\ c\ Q,A$
  shows $\Gamma,\{\} \vdash_{t/F} P\ c\ Q,A$
  using *MGT*
  apply (*rule ConseqMGT*)
  apply (*insert valid*)
  apply (*auto simp add*: *validt-def valid-def intro*!: *final-notinI*)
  done

lemma *conseq-extract-state-indep-prop*:
  assumes *state-indep-prop*:$\forall s \in P.\ R$
  assumes *to-show*: $R \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
  shows $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
  apply (*rule Conseq*)
  apply (*clarify*)
  apply (*rule-tac x*=*P* **in** *exI*)
  apply (*rule-tac x*=*Q* **in** *exI*)
  apply (*rule-tac x*=*A* **in** *exI*)
  using *state-indep-prop to-show*
  by *blast*

lemma *MGT-lemma*:
  assumes *MGT-Calls*:
    $\forall p \in dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F}$
      $\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
        $\Gamma\vdash(Call\ p)\downarrow Normal\ s\}$
          $(Call\ p)$

$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
shows $\bigwedge Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash c\downarrow Normal\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*induct c*)
case *Skip*
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Skip,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Skip \downarrow Normal\ s\}$
Skip
$\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Abrupt$
$t\}$
by (*rule hoaret.Skip* [*THEN conseqPre*])
(*auto elim*: *exec-elim-cases simp add*: *final-notin-def*
intro: *exec.intros terminates.intros*)
next
case (*Basic f*)
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Basic\ f,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Basic\ f \downarrow Normal\ s\}$
Basic f
$\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Basic* [*THEN conseqPre*])
(*auto elim*: *exec-elim-cases simp add*: *final-notin-def*
intro: *exec.intros terminates.intros*)
next
case (*Spec r*)
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

$\Gamma\vdash Spec\ r \downarrow Normal\ s\}$
Spec r
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoaret.Spec* [*THEN conseqPre*])
apply (*clarsimp simp add*: *final-notin-def*)
apply (*case-tac* $\exists\, t.\ (Z,t) \in r$)
apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
done
next
case (*Seq c1 c2*)
have *hyp-c1*: $\forall\, Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash c1\downarrow Normal\ s\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

using *Seq.hyps* **by** *iprover*
have *hyp-c2*: $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash c2\downarrow Normal\ s\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Seq.hyps* **by** *iprover*
from *hyp-c1*
have $\Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Seq\ c1\ c2\downarrow Normal\ s\}\ c1$
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t \wedge \Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash c2\downarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*auto dest*: *Seq-NoFaultStuckD1* [*simplified*] *Seq-NoFaultStuckD2* [*simplified*]
elim: *terminates-Normal-elim-cases*
intro: *exec.intros*)
thus $\Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Seq\ c1\ c2\downarrow Normal\ s\}$
$Seq\ c1\ c2$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule hoaret.Seq*)
show $\Gamma,\Theta\vdash_{t/F}\ \{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t \wedge$
$\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c2 \downarrow Normal\ t\}$
$c2$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
fix *r t*
assume $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ r\ \Gamma\vdash\langle c2,Normal\ r\rangle \Rightarrow Normal\ t$
then show $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*rule exec.intros*)
next
fix *r t*
assume $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ r\ \Gamma\vdash\langle c2,Normal\ r\rangle \Rightarrow Abrupt\ t$
then show $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*rule exec.intros*)
qed
qed
next
case (*Cond b c1 c2*)
have $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

Γ⊢*c1*↓*Normal s*}
c1
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Abrupt t*}
using *Cond.hyps* **by** *iprover*
hence Γ,Θ $\vdash_{t/F}$ ({*s*. *s*=*Z* ∧ Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢(*Cond b c1 c2*)↓*Normal s*}∩*b*)
c1
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondTrue*
elim: *terminates-Normal-elim-cases*)
moreover
have ∀ *Z*. Γ,Θ$\vdash_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*c2*↓*Normal s*}
c2
{*t*. Γ⊢⟨*c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
using *Cond.hyps* **by** *iprover*
hence Γ,Θ$\vdash_{t/F}$ ({*s*. *s*=*Z* ∧ Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢(*Cond b c1 c2*)↓*Normal s*}∩−*b*)
c2
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondFalse*
elim: *terminates-Normal-elim-cases*)
ultimately
show Γ,Θ$\vdash_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢(*Cond b c1 c2*)↓*Normal s*}
(*Cond b c1 c2*)
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
by (*rule hoaret.Cond*)
next
case (*While b c*)
let *?unroll* = ({(*s*,*t*). *s*∈*b* ∧ Γ⊢⟨*c*,*Normal s*⟩ ⇒ *Normal t*})*
let *?P'* = λ*Z*. {*t*. (*Z*,*t*)∈*?unroll* ∧
(∀ *e*. (*Z*,*e*)∈*?unroll* ⟶ *e*∈*b*
⟶ Γ⊢⟨*c*,*Normal e*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
(∀ *u*. Γ⊢⟨*c*,*Normal e*⟩ ⇒*Abrupt u* ⟶
Γ⊢⟨*While b c*,*Normal Z*⟩ ⇒ *Abrupt u*)) ∧
Γ⊢(*While b c*)↓*Normal t*}
let *?A* = λ*Z*. {*t*. Γ⊢⟨*While b c*,*Normal Z*⟩ ⇒ *Abrupt t*}

let $?r = \{(t,s).\ \Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ s \wedge s\in b \wedge$
$\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\}$
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F))$
$\wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ s\}$
$(\mathit{While}\ b\ c)$
$\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [**where** $?P'=\lambda\ Z.\ ?P'\ Z$
and $?Q'=\lambda\ Z.\ ?P'\ Z \cap -\ b$])
have *wf-r*: *wf ?r* **by** (*rule wf-terminates-while*)
show $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F}\ (?P'\ Z)\ (\mathit{While}\ b\ c)\ (?P'\ Z \cap -\ b),(?A\ Z)$
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix Z
from *While*
have *hyp-c*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}$
$(-F)) \wedge$
$\Gamma\vdash c\downarrow\mathit{Normal}\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$ **by** *iprover*
show $\forall\sigma.\ \Gamma,\Theta\vdash_{t/F}\ (\{\sigma\} \cap\ ?P'\ Z\ \cap b)\ c$
$(\{t.\ (t,\ \sigma) \in\ ?r\} \cap\ ?P'\ Z),(?A\ Z)$
proof (*rule allI*, *rule ConseqMGT* [*OF hyp-c*])
fix $\sigma\ s$
assume $s\in \{\sigma\} \cap$
$\{t.\ (Z,\ t) \in\ ?\mathit{unroll} \wedge$
$(\forall e.\ (Z,e)\in ?\mathit{unroll} \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ t\}$
$\cap\ b$
then obtain
s-eq-σ: $s=\sigma$ **and**
Z-s-unroll: $(Z,s) \in\ ?\mathit{unroll}$ **and**
noabort: $\forall e.\ (Z,e)\in ?\mathit{unroll} \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)$ **and**
while-term: $\Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ s$ **and**
s-in-b: $s\in b$
by *blast*
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$\Gamma\vdash c\downarrow\mathit{Normal}\ t\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\} \longrightarrow$
$t \in \{t.\ (t,\sigma) \in\ ?r\} \cap$
$\{t.\ (Z,\ t) \in\ ?\mathit{unroll} \wedge$
$(\forall e.\ (Z,e)\in ?\mathit{unroll} \longrightarrow\ e\in b$

```
                    ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                      (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                          Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u)) ∧
                Γ⊢(While b c)↓Normal t}) ∧
        (∀ t. t ∈ {t. Γ⊢⟨c,Normal s⟩ ⇒ Abrupt t} ⟶
            t ∈ {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t})
        (is ?C1 ∧ ?C2 ∧ ?C3)
      proof (intro conjI)
        from Z-s-unroll noabort s-in-b while-term show ?C1
          by (blast elim: terminates-Normal-elim-cases)
      next
        {
          fix t
          assume s-t: Γ⊢⟨c,Normal s⟩ ⇒ Normal t
          with s-eq-σ while-term s-in-b have (t,σ) ∈ ?r
            by blast
          moreover
          from Z-s-unroll s-t s-in-b
          have (Z, t) ∈ ?unroll
            by (blast intro: rtrancl-into-rtrancl)
          moreover from while-term s-t s-in-b
          have Γ⊢(While b c)↓Normal t
            by (blast elim: terminates-Normal-elim-cases)
          moreover note noabort
          ultimately
          have (t,σ) ∈ ?r ∧ (Z, t) ∈ ?unroll ∧
                (∀ e. (Z,e)∈?unroll ⟶ e∈b
                    ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                       (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                           Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u)) ∧
                Γ⊢(While b c)↓Normal t
            by iprover
        }
        then show ?C2 by blast
      next
        {
          fix t
          assume s-t: Γ⊢⟨c,Normal s⟩ ⇒ Abrupt t
          from Z-s-unroll noabort s-t s-in-b
          have Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t
            by blast
        } thus ?C3 by simp
      qed
    qed
  qed
next
  fix s
  assume P: s ∈ {s. s=Z ∧ Γ⊢⟨While b c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F)) ∧
```

```
              Γ⊢ While b c ↓ Normal s}
hence WhileNoFault: Γ⊢⟨While b c,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
  by auto
show s ∈ ?P′ s ∧
 (∀ t. t∈(?P′ s ∩ − b)⟶
     t∈{t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t})∧
 (∀ t. t∈?A s ⟶ t∈?A Z)
proof (intro conjI)
  {
    fix e
    assume (Z,e) ∈ ?unroll e ∈ b
    from this WhileNoFault
    have Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
           (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
               Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u) (is ?Prop Z e)
    proof (induct rule: converse-rtrancl-induct [consumes 1])
      assume e-in-b: e ∈ b
       assume WhileNoFault: Γ⊢⟨While b c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))
      with e-in-b WhileNoFault
      have cNoFault: Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
        by (auto simp add: final-notin-def intro: exec.intros)
      moreover
      {
        fix u assume Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u
        with e-in-b have Γ⊢⟨While b c,Normal e⟩ ⇒ Abrupt u
          by (blast intro: exec.intros)
      }
      ultimately
      show ?Prop e e
        by iprover
    next
      fix Z r
      assume e-in-b: e∈b
       assume WhileNoFault: Γ⊢⟨While b c,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F))
      assume hyp: ⟦e∈b;Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))⟧
                 ⟹ ?Prop r e
      assume Z-r:
        (Z, r) ∈ {(Z, r). Z ∈ b ∧ Γ⊢⟨c,Normal Z⟩ ⇒ Normal r}
      with WhileNoFault
      have Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
        by (auto simp add: final-notin-def intro: exec.intros)
      from hyp [OF e-in-b this] obtain
        cNoFault: Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) and
        Abrupt-r: ∀ u. Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u ⟶
                     Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u
        by simp
```

```
      {
       fix u assume Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u
       with Abrupt-r have Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u by simp
       moreover from  Z-r obtain
         Z ∈ b  Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
         by simp
       ultimately have Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u
         by (blast intro: exec.intros)
      }
      with cNoFault show ?Prop Z e
       by iprover
    qed
  }
  with P show s ∈ ?P' s
    by blast
next
  {
    fix t
    assume termination: t ∉ b
    assume (Z, t) ∈ ?unroll
    hence Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
    proof (induct rule: converse-rtrancl-induct [consumes 1])
      from termination
      show Γ⊢⟨While b c,Normal t⟩ ⇒ Normal t
       by (blast intro: exec.WhileFalse)
    next
      fix Z r
      assume first-body:
            (Z, r) ∈ {(s, t). s ∈ b ∧ Γ⊢⟨c,Normal s⟩ ⇒ Normal t}
      assume (r, t) ∈ ?unroll
      assume rest-loop: Γ⊢⟨While b c, Normal r⟩ ⇒ Normal t
      show Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
      proof −
       from first-body obtain
         Z ∈ b Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
         by fast
       moreover
       from rest-loop have
         Γ⊢⟨While b c,Normal r⟩ ⇒ Normal t
         by fast
       ultimately show Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
         by (rule exec.WhileTrue)
      qed
    qed
  }
  with P
  show (∀ t. t∈(?P' s ∩ − b)
       ⟶t∈{t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t})
    by blast
```

next
from P **show** $\forall t.\ t{\in}?A\ s \longrightarrow t{\in}?A\ Z$
by *simp*
qed
qed
next
case (*Call p*)
from *noStuck-Call*
have $\forall s \in \{s.\ s{=}Z \wedge \Gamma{\vdash}\langle Call\ p,Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma{\vdash}Call\ p{\downarrow}\ Normal\ s\}.$
$p \in dom\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show *?case*
proof (*rule conseq-extract-state-indep-prop*)
assume *p-defined*: $p \in dom\ \Gamma$
with *MGT-Calls* **show**
$\Gamma,\Theta{\vdash}_{t/F}\ \{s.\ s{=}Z \wedge$
$\Gamma{\vdash}\langle Call\ p\ ,Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F)){\wedge}$
$\Gamma{\vdash}Call\ \ p{\downarrow}Normal\ s\}$
$(Call\ p)$
$\{t.\ \Gamma{\vdash}\langle Call\ p,Normal\ Z\rangle \Rightarrow\ Normal\ t\},$
$\{t.\ \Gamma{\vdash}\langle Call\ p,Normal\ Z\rangle \Rightarrow\ Abrupt\ t\}$
by (*auto*)
qed
next
case (*DynCom c*)
have *hyp*:
$\bigwedge s'.\ \forall Z.\ \Gamma,\Theta{\vdash}_{t/F}\ \{s.\ s = Z \wedge \Gamma{\vdash}\langle c\ s',Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma{\vdash}c\ s'{\downarrow}Normal\ s\}\ c\ s'$
$\{t.\ \Gamma{\vdash}\langle c\ s',Normal\ Z\rangle \Rightarrow\ Normal\ t\},\{t.\ \Gamma{\vdash}\langle c\ s',Normal\ Z\rangle \Rightarrow\ Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta{\vdash}_{t/F}\ \{s.\ s = Z \wedge \Gamma{\vdash}\langle DynCom\ c,Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma{\vdash}DynCom\ c{\downarrow}Normal\ s\}$
$(c\ Z)$
$\{t.\ \Gamma{\vdash}\langle DynCom\ c,Normal\ Z\rangle \Rightarrow\ Normal\ t\},\{t.\ \Gamma{\vdash}\langle DynCom\ c,Normal\ Z\rangle$
$\Rightarrow\ Abrupt\ t\}$
by (*rule ConseqMGT* [*OF hyp*])
(*fastforce simp add*: *final-notin-def intro*: *exec.intros*
elim: *terminates-Normal-elim-cases*)
show $\Gamma,\Theta{\vdash}_{t/F}\ \{s.\ s{=}Z \wedge \Gamma{\vdash}\langle DynCom\ c,Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma{\vdash}DynCom\ c{\downarrow}Normal\ s\}$
$DynCom\ c$
$\{t.\ \Gamma{\vdash}\langle DynCom\ c,Normal\ Z\rangle \Rightarrow\ Normal\ t\},$
$\{t.\ \Gamma{\vdash}\langle DynCom\ c,Normal\ Z\rangle \Rightarrow\ Abrupt\ t\}$
apply (*rule hoaret.DynCom*)
apply (*clarsimp*)

```
    apply (rule hyp' [simplified])
    done
next
  case (Guard f g c)
  have hyp-c: ∀ Z. Γ,Θ⊢t/F {s. s=Z ∧ Γ⊢⟨c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F)) ∧
                              Γ⊢c↓Normal s}
                      c
                    {t. Γ⊢⟨c,Normal Z⟩ ⇒ Normal t},
                    {t. Γ⊢⟨c,Normal Z⟩ ⇒ Abrupt t}
    using Guard by iprover
  show Γ,Θ⊢t/F {s. s = Z ∧ Γ⊢⟨Guard f g c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘
(−F)) ∧
                    Γ⊢Guard f g c↓ Normal s}
                 Guard f g c
               {t. Γ⊢⟨Guard f g c ,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Abrupt t}
  proof (cases f ∈ F)
    case True
    from hyp-c
    have Γ,Θ⊢t/F (g ∩ {s. s=Z ∧
                    Γ⊢⟨Guard f g c,Normal s⟩ ⇒∉({Stuck}∪ Fault ‘ (−F))∧
                    Γ⊢Guard f g c↓ Normal s})
                  c
                {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Normal t},
                {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Abrupt t}
      apply (rule ConseqMGT)
      apply (insert True)
      apply  (auto simp add: final-notin-def intro: exec.intros
                elim: terminates-Normal-elim-cases)
      done
    from True this
    show ?thesis
      by (rule conseqPre [OF Guarantee]) auto
  next
    case False
    from hyp-c
    have Γ,Θ⊢t/F (g ∩ {s. s ∈ g ∧ s=Z ∧
                    Γ⊢⟨Guard f g c,Normal s⟩ ⇒∉({Stuck}∪ Fault ‘ (−F))∧
                    Γ⊢Guard f g c↓ Normal s} )
                  c
                {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Normal t},
                {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Abrupt t}
      apply (rule ConseqMGT)
      apply clarify
      apply (frule Guard-noFaultStuckD [OF - False])
      apply  (auto simp add: final-notin-def intro: exec.intros
                elim: terminates-Normal-elim-cases)
      done
```

then show *?thesis*
apply (*rule conseqPre* [*OF hoaret.Guard*])
apply *clarify*
apply (*frule Guard-noFaultStuckD* [*OF - False*])
apply *auto*
done
qed
next
case *Throw*
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
$\wedge$
$\Gamma\vdash \mathit{Throw} \downarrow \mathit{Normal}\ s\}$
Throw
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule conseqPre* [*OF hoaret.Throw*])
(*blast intro*: *exec.intros terminates.intros*)
next
case (*Catch* c_1 c_2)
have $\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
$\wedge$
$\Gamma\vdash c_1 \downarrow \mathit{Normal}\ s\}$
c_1
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}$
$(-F)) \wedge$
$\Gamma\vdash \mathit{Catch}\ c_1\ c_2 \downarrow \mathit{Normal}\ s\}$
c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t \wedge \Gamma\vdash c_2 \downarrow \mathit{Normal}\ t \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros terminates.intros*
elim: *terminates-Normal-elim-cases*
simp add: *final-notin-def*)
moreover
have
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F)) \wedge$
$\Gamma\vdash c_2 \downarrow \mathit{Normal}\ s\}\ c_2$
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{t/F} \{s.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ s \wedge \Gamma\vdash c_2 \downarrow \mathit{Normal}\ s \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
c_2
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$

by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros terminates.intros*
simp add: *noFault-def'*)
ultimately
show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Catch\ c_1\ c_2 \downarrow Normal\ s\}$
$Catch\ c_1\ c_2$
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Catch*)
qed

lemma *Call-lemma'*:
assumes *Call-hyp*:
$\forall\, q\in dom\ \Gamma.\ \forall\, Z.\ \Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ q\downarrow Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge c \in redexes\ c')\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*induct c*)
case *Skip*
show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Skip,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge Skip \in redexes\ c')\}$
$Skip$
$\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Skip,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Skip* [*THEN conseqPre*]) (*blast intro*: *exec.Skip*)
next
case (*Basic f*)
show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle Basic\ f,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge$
$Basic\ f \in redexes\ c')\}$
$Basic\ f$
$\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Basic\ f,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Basic* [*THEN conseqPre*]) (*blast intro*: *exec.Basic*)

next
  case (*Spec r*)
  show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \land \Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$

$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land$
$Spec\ r \in redexes\ c')\}$
Spec r
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    apply (*rule hoaret.Spec* [*THEN conseqPre*])
    apply (*clarsimp*)
    apply (*case-tac* $\exists\, t.\ (Z,t) \in r$)
    apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
    done
next
  case (*Seq c1 c2*)
  have *hyp-c1*:
    $\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \land \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land c1 \in redexes\ c')\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *Seq.hyps* **by** *iprover*
  have *hyp-c2*:
    $\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \land \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land c2 \in redexes\ c')\}$
c2
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    using *Seq.hyps* (*2*) **by** *iprover*
  have *c1*: $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \land \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \land$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land$
$Seq\ c1\ c2 \in redexes\ c')\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t \land$
$\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ t) \land$
$c2 \in redexes\ c')\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  proof (*rule ConseqMGT* [*OF hyp-c1*],*clarify*,*safe*)
    assume $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
    thus $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
      by (*blast dest*: *Seq-NoFaultStuckD1*)

next
  fix c'
  assume *steps-c'*: $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal Z*)
  assume *red*: *Seq c1 c2* $\in$ *redexes* c'
  from *redexes-subset* [*OF red*] *steps-c'*
  show $\exists\, c'$. $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal Z*) $\wedge$ *c1* $\in$ *redexes* c'
    by (*auto iff*: *root-in-redexes*)
next
  fix *t*
  assume $\Gamma\vdash\langle$*Seq c1 c2*,*Normal Z*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ` $(-F))$
    $\Gamma\vdash\langle$*c1*,*Normal Z*$\rangle \Rightarrow$ *Normal t*
  thus $\Gamma\vdash\langle$*c2*,*Normal t*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ` $(-F))$
    by (*blast dest*: *Seq-NoFaultStuckD2*)
next
  fix c' *t*
  assume *steps-c'*: $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal Z*)
  assume *red*: *Seq c1 c2* $\in$ *redexes* c'
  assume *exec-c1*: $\Gamma\vdash$ $\langle$*c1*,*Normal Z*$\rangle \Rightarrow$ *Normal t*
  show $\exists\, c'$. $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal t*) $\wedge$ *c2* $\in$ *redexes* c'
  proof −
    note *steps-c'*
    also
    from *exec-impl-steps-Normal* [*OF exec-c1*]
    have $\Gamma\vdash$ (*c1*, *Normal Z*) $\rightarrow^*$ (*Skip*, *Normal t*)**.**
    from *steps-redexes-Seq* [*OF this red*]
    obtain c'' **where**
      steps-c'': $\Gamma\vdash$ (c', *Normal Z*) $\rightarrow^*$ (c'', *Normal t*) **and**
      Skip: *Seq Skip c2* $\in$ *redexes* c''
      by *blast*
    note *steps-c''*
    also
    have *step-Skip*: $\Gamma\vdash$ (*Seq Skip c2*,*Normal t*) $\rightarrow$ (*c2*,*Normal t*)
      by (*rule step.SeqSkip*)
    from *step-redexes* [*OF step-Skip Skip*]
    obtain c''' **where**
      step-c''': $\Gamma\vdash$ (c'', *Normal t*) $\rightarrow$ (c''', *Normal t*) **and**
      c2: *c2* $\in$ *redexes* c'''
      by *blast*
    note *step-c'''*
    finally show *?thesis*
      using *c2*
      by *blast*
  qed
next
  fix *t*
  assume $\Gamma\vdash\langle$*c1*,*Normal Z*$\rangle \Rightarrow$ *Abrupt t*
  thus $\Gamma\vdash\langle$*Seq c1 c2*,*Normal Z*$\rangle \Rightarrow$ *Abrupt t*
    by (*blast intro*: *exec.intros*)
qed

show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge Seq\ c1\ c2 \in redexes$
$c')\}$
$Seq\ c1\ c2$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Seq* [*OF c1 ConseqMGT* [*OF hyp-c2*]])
(*blast intro*: *exec.intros*)
next
case (*Cond b c1 c2*)
have *hyp-c1*:
$\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge c1 \in redexes\ c')\}$
$c1$
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
have
$\Gamma,\Theta\vdash_{t/F}$ $(\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge$
$Cond\ b\ c1\ c2 \in redexes\ c')\}$
$\cap\ b)$
$c1$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*safe*)
assume $Z \in b$ $\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.CondTrue*)
next
fix c'
assume *b*: $Z \in b$
assume *steps-c'*: $\Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z)$
assume *redex-c'*: $Cond\ b\ c1\ c2 \in redexes\ c'$
show $\exists\, c'.\ \Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z) \wedge c1 \in redexes\ c'$
proof −
note *steps-c'*
also
from *b*
have $\Gamma\vdash(Cond\ b\ c1\ c2,\ Normal\ Z) \rightarrow (c1,\ Normal\ Z)$
by (*rule step.CondTrue*)
from *step-redexes* [*OF this redex-c'*] **obtain** c'' **where**
step-c'': $\Gamma\vdash (c',\ Normal\ Z) \rightarrow (c'',\ Normal\ Z)$ **and**
c1: $c1 \in redexes\ c''$

by *blast*
note *step-c''*
finally show *?thesis*
using *c1*
by *blast*
qed
next
fix *t* **assume** $Z \in b$ $\Gamma \vdash \langle c1, Normal\ Z\rangle \Rightarrow Normal\ t$
thus $\Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Normal\ t$
by (*blast intro*: *exec.CondTrue*)
next
fix *t* **assume** $Z \in b$ $\Gamma \vdash \langle c1, Normal\ Z\rangle \Rightarrow Abrupt\ t$
thus $\Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*blast intro*: *exec.CondTrue*)
qed
moreover
have *hyp-c2*:
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} \{s.\ s=Z \wedge \Gamma \vdash \langle c2, Normal\ s\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge c2 \in redexes\ c')\}$
$c2$
$\{t.\ \Gamma \vdash \langle c2, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma \vdash \langle c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
have
$\Gamma,\Theta \vdash_{t/F} (\{s.\ s=Z \wedge \Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ s\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge$
$Cond\ b\ c1\ c2 \in redexes\ c')\}$
$\cap -b)$
$c2$
$\{t.\ \Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
assume $Z \notin b$ $\Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma \vdash \langle c2, Normal\ Z\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.CondFalse*)
next
fix c'
assume *b*: $Z \notin b$
assume *steps-c'*: $\Gamma \vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
assume *redex-c'*: $Cond\ b\ c1\ c2 \in redexes\ c'$
show $\exists c'.\ \Gamma \vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z) \wedge c2 \in redexes\ c'$
proof −
note *steps-c'*
also
from *b*
have $\Gamma \vdash (Cond\ b\ c1\ c2,\ Normal\ Z) \rightarrow (c2,\ Normal\ Z)$

by (*rule step.CondFalse*)
from *step-redexes* [*OF this redex-c'*] **obtain** c'' **where**
step-c'': $\Gamma\vdash$ (c', *Normal Z*) $\rightarrow$ (c'', *Normal Z*) **and**
c1: *c2* $\in$ *redexes* c''
by *blast*
note *step-c''*
finally show *?thesis*
using *c1*
by *blast*
qed
next
fix *t* **assume** $Z \notin b$ $\Gamma\vdash\langle c2, Normal\ Z\rangle \Rightarrow Normal\ t$
thus $\Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Normal\ t$
by (*blast intro*: *exec.CondFalse*)
next
fix *t* **assume** $Z \notin b$ $\Gamma\vdash\langle c2, Normal\ Z\rangle \Rightarrow Abrupt\ t$
thus $\Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*blast intro*: *exec.CondFalse*)
qed
ultimately
show
$\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists\,c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge$
$Cond\ b\ c1\ c2 \in redexes\ c')\}$
$(Cond\ b\ c1\ c2)$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Cond*)
next
case (*While b c*)
let $?unroll = (\{(s,t).\ s{\in}b \wedge \Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow Normal\ t\})^{*}$
let $?P' = \lambda Z.\ \{t.\ (Z,t){\in}?unroll \wedge$
$(\forall\,e.\ (Z,e){\in}?unroll \longrightarrow e{\in}b$
$\longrightarrow \Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall\,u.\ \Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists\,c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+}$
$(c', Normal\ t) \wedge While\ b\ c \in redexes\ c')\}$
let $?A = \lambda Z.\ \{t.\ \Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
let $?r = \{(t,s).\ \Gamma\vdash(While\ b\ c){\downarrow}Normal\ s \wedge s{\in}b \wedge$
$\Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow Normal\ t\}$
show $\Gamma,\Theta\vdash_{t/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists\,c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma)\rightarrow^{+}(c', Normal\ s) \wedge While\ b\ c \in redexes\ c')\}$
$(While\ b\ c)$

$\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [**where** $?P'=\lambda\ Z.\ ?P'\ Z$
and $?Q'=\lambda\ Z.\ ?P'\ Z \cap -\ b$])
have *wf-r*: *wf ?r* **by** (*rule wf-terminates-while*)
show $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F}$ $(?P'\ Z)$ $(\mathit{While}\ b\ c)$ $(?P'\ Z \cap -\ b),(?A\ Z)$
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix *Z*
from *While*
have *hyp-c*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ p\!\downarrow\!\mathit{Normal}\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c',\mathit{Normal}\ s) \wedge c \in \mathit{redexes}\ c')\}$
c
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$ **by** *iprover*
show $\forall\sigma.\ \Gamma,\Theta\vdash_{t/F}$ $(\{\sigma\} \cap ?P'\ Z\ \cap b)$ c
$(\{t.\ (t,\ \sigma) \in ?r\} \cap ?P'\ Z),(?A\ Z)$
proof (*rule allI*, *rule ConseqMGT* [*OF hyp-c*])
fix τ s
assume *asm*: $s\in \{\tau\} \cap$
$\{t.\ (Z,\ t) \in ?\mathit{unroll} \wedge$
$(\forall\, e.\ (Z,e)\in ?\mathit{unroll} \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash \mathit{Call}\ p\!\downarrow\ \mathit{Normal}\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+}$
$(c',\mathit{Normal}\ t) \wedge \mathit{While}\ b\ c \in \mathit{redexes}\ c')\}$
$\cap\ b$
then obtain c' **where**
s-eq-τ: $s=\tau$ **and**
Z-s-unroll: $(Z,s) \in ?\mathit{unroll}$ **and**
noabort: $\forall\, e.\ (Z,e)\in ?\mathit{unroll} \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{‘}\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)$ **and**
termi: $\Gamma\vdash \mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma$ **and**
reach: $\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c',\mathit{Normal}\ s)$ **and**
red-c': $\mathit{While}\ b\ c \in \mathit{redexes}\ c'$ **and**
s-in-b: $s\in b$
by *blast*
obtain c'' **where**
reach-c: $\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c'',\mathit{Normal}\ s)$
$\mathit{Seq}\ c\ (\mathit{While}\ b\ c) \in \mathit{redexes}\ c''$
proof −
note *reach*
also from *s-in-b*
have $\Gamma\vdash(\mathit{While}\ b\ c,\mathit{Normal}\ s) \rightarrow (\mathit{Seq}\ c\ (\mathit{While}\ b\ c),\mathit{Normal}\ s)$

by (*rule step.WhileTrue*)
from *step-redexes* [*OF this red-c'*] **obtain** c'' **where**
step: $\Gamma\vdash$ (c', *Normal s*) $\rightarrow$ (c'', *Normal s*) **and**
red-c'': *Seq c* (*While b c*) $\in$ *redexes* c''
by *blast*
note *step*
finally
show *?thesis*
using *red-c''*
by (*blast intro*: *that*)
qed
from *reach termi*
have $\Gamma\vdash c' \downarrow$ *Normal s*
by (*rule steps-preserves-termination'*)
from *redexes-preserves-termination* [*OF this red-c'*]
have *termi-while*: $\Gamma\vdash$ *While b c* $\downarrow$ *Normal s* **.**
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c',\mathit{Normal}\ t) \wedge c \in \mathit{redexes}\ c')\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\} \longrightarrow$
$t \in \{t.\ (t,\tau) \in\ ?r\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow\ e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash \mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c',\mathit{Normal}\ t) \wedge$
$\mathit{While}\ b\ c \in \mathit{redexes}\ c')\}) \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Abrupt}\ t\} \longrightarrow$
$t \in \{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\})$
(**is** *?C1* $\wedge$ *?C2* $\wedge$ *?C3*)
proof (*intro conjI*)
from *Z-s-unroll noabort s-in-b termi reach-c* **show** *?C1*
apply *clarsimp*
apply (*drule redexes-subset*)
apply *simp*
apply (*blast intro*: *root-in-redexes*)
done
next
{
fix *t*
assume *s-t*: $\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t$
with *s-eq-*τ *termi-while s-in-b* **have** $(t,\tau) \in\ ?r$
by *blast*
moreover
from *Z-s-unroll s-t s-in-b*
have $(Z,\ t) \in\ ?unroll$
by (*blast intro*: *rtrancl-into-rtrancl*)

```
moreover
obtain c'' where
  reach-c'': Γ⊢(Call p,Normal σ) →⁺ (c'',Normal t)
          (While b c) ∈ redexes c''
proof –
  note reach-c (1)
  also from s-in-b
  have Γ⊢(While b c,Normal s)→ (Seq c (While b c),Normal s)
    by (rule step.WhileTrue)
  have Γ⊢ (Seq c (While b c), Normal s) →⁺
          (While b c, Normal t)
  proof –
    from exec-impl-steps-Normal [OF s-t]
    have Γ⊢ (c, Normal s) →* (Skip, Normal t).
    hence Γ⊢ (Seq c (While b c), Normal s) →*
            (Seq Skip (While b c), Normal t)
      by (rule SeqSteps) auto
    moreover
    have Γ⊢(Seq Skip (While b c), Normal t)→(While b c, Normal t)
      by (rule step.SeqSkip)
    ultimately show ?thesis by (rule rtranclp-into-tranclp1)
  qed
  from steps-redexes' [OF this reach-c (2)]
  obtain c''' where
    step: Γ⊢ (c'', Normal s) →⁺ (c''', Normal t) and
    red-c'': While b c ∈ redexes c'''
    by blast
  note step
  finally
  show ?thesis
    using red-c''
    by (blast intro: that)
qed
moreover note noabort termi
ultimately
have (t,τ) ∈ ?r ∧ (Z, t) ∈ ?unroll ∧
      (∀ e. (Z,e)∈?unroll ⟶ e∈b
            ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                      Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u)) ∧
      Γ⊢Call p ↓ Normal σ ∧
        (∃ c'. Γ⊢(Call p,Normal σ) →⁺ (c', Normal t) ∧
                While b c ∈ redexes c')
  by iprover
}
then show ?C2 by blast
next
{
  fix t
```

assume *s-t*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Abrupt\ t$
from *Z-s-unroll noabort s-t s-in-b*
have $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by *blast*
} **thus** *?C3* **by** *simp*
qed
qed
qed
next
fix *s*
assume *P*: $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge$
$While\ b\ c \in redexes\ c')\}$
hence *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by *auto*
show $s \in ?P'\ s \wedge$
$(\forall t.\ t\in(?P'\ s \cap -\ b)\longrightarrow$
$t\in\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\})\wedge$
$(\forall t.\ t\in ?A\ s \longrightarrow t\in ?A\ Z)$
proof (*intro conjI*)
{
fix *e*
assume $(Z,e) \in ?unroll\ e \in b$
from *this WhileNoFault*
have $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)$ (**is** *?Prop Z e*)
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
assume *e-in-b*: $e \in b$
assume *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
with *e-in-b WhileNoFault*
have *cNoFault*: $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
moreover
{
fix *u* **assume** $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u$
with *e-in-b* **have** $\Gamma\vdash\langle While\ b\ c,Normal\ e\rangle \Rightarrow Abrupt\ u$
by (*blast intro*: *exec.intros*)
}
ultimately
show *?Prop e e*
by *iprover*
next
fix *Z r*
assume *e-in-b*: $e\in b$
assume *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$

```
(−F))
      assume hyp: ⟦e∈b;Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))⟧
                 ⟹ ?Prop r e
      assume Z-r:
        (Z, r) ∈ {(Z, r). Z ∈ b ∧ Γ⊢⟨c,Normal Z⟩ ⇒ Normal r}
      with WhileNoFault
      have Γ⊢⟨While b c,Normal r⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
        by (auto simp add: final-notin-def intro: exec.intros)
      from hyp [OF e-in-b this] obtain
        cNoFault: Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) and
        Abrupt-r: ∀ u. Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u ⟶
                     Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u
        by simp

       {
        fix u assume Γ⊢⟨c,Normal e⟩ ⇒ Abrupt u
        with Abrupt-r have Γ⊢⟨While b c,Normal r⟩ ⇒ Abrupt u by simp
        moreover from  Z-r obtain
          Z ∈ b  Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
          by simp
        ultimately have Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u
          by (blast intro: exec.intros)
      }
      with cNoFault show ?Prop Z e
        by iprover
    qed
  }
  with P show s ∈ ?P' s
    by blast
next
  {
    fix t
    assume termination: t ∉ b
    assume (Z, t) ∈ ?unroll
    hence Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
    proof (induct rule: converse-rtrancl-induct [consumes 1])
      from termination
      show Γ⊢⟨While b c,Normal t⟩ ⇒ Normal t
        by (blast intro: exec.WhileFalse)
    next
      fix Z r
      assume first-body:
            (Z, r) ∈ {(s, t). s ∈ b ∧ Γ⊢⟨c,Normal s⟩ ⇒ Normal t}
      assume (r, t) ∈ ?unroll
      assume rest-loop: Γ⊢⟨While b c, Normal r⟩ ⇒ Normal t
      show Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t
      proof −
        from first-body obtain
          Z ∈ b Γ⊢⟨c,Normal Z⟩ ⇒ Normal r
```

by *fast*
moreover
from *rest-loop* **have**
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ r\rangle \Rightarrow \mathit{Normal}\ t$
by *fast*
ultimately show $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
by (*rule exec.WhileTrue*)
qed
qed
}
with P
show $\forall\, t.\ t{\in}(?P'\ s \cap -\ b)$
$\longrightarrow t{\in}\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$
by *blast*
next
from P **show** $\forall\, t.\ t{\in}?A\ s \longrightarrow t{\in}?A\ Z$
by *simp*
qed
qed
next
case (*Call q*)
let $?P = \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q\ ,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ p{\downarrow}\mathit{Normal}\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^{+} (c',\mathit{Normal}\ s) \wedge \mathit{Call}\ q \in \mathit{redexes}\ c')\}$
from *noStuck-Call*
have $\forall\, s \in\ ?P.\ q \in \mathit{dom}\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show *?case*
proof (*rule conseq-extract-state-indep-prop*)
assume *q-defined*: $q \in \mathit{dom}\ \Gamma$
from *Call-hyp* **have**
$\forall\, q{\in}\mathit{dom}\ \Gamma.\ \forall\, Z.$
$\Gamma,\Theta\vdash_{t/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ q{\downarrow}\mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*simp add*: *exec-Call-body' noFaultStuck-Call-body'* [*simplified*]
terminates-Normal-Call-body)
from *Call-hyp q-defined* **have** *Call-hyp'*:
$\forall\, Z.\ \Gamma,\Theta \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash \mathit{Call}\ q{\downarrow}\mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by *auto*
show
$\Gamma,\Theta\vdash_{t/F}\ ?P$

$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q\ ,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF Call-hyp'*],*safe*)
fix c'
assume *termi*: $\Gamma\vdash Call\ p \downarrow Normal\ \sigma$
assume *steps-c'*: $\Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z)$
assume *red-c'*: $Call\ q \in redexes\ c'$
show $\Gamma\vdash Call\ q \downarrow Normal\ Z$
proof −
from *steps-preserves-termination'* [*OF steps-c' termi*]
have $\Gamma\vdash c' \downarrow Normal\ Z$ **.**
from *redexes-preserves-termination* [*OF this red-c'*]
show *?thesis* **.**
qed
next
fix c'
assume *termi*: $\Gamma\vdash Call\ p \downarrow Normal\ \sigma$
assume *steps-c'*: $\Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z)$
assume *red-c'*: $Call\ q \in redexes\ c'$
from *redex-redexes* [*OF this*]
have $redex\ c' = Call\ q$
by *auto*
with *termi steps-c'*
show $((Z,\ q),\ \sigma,\ p) \in termi\text{-}call\text{-}steps\ \Gamma$
by (*auto simp add*: *termi-call-steps-def*)
qed
qed
next
case (*DynCom c*)
have *hyp*:
$\bigwedge s'.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s = Z \wedge \Gamma\vdash\langle c\ s',Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge c\ s' \in redexes\ c')\}$
$(c\ s')$
$\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge DynCom\ c \in redexes$
$c')\}$
$(c\ Z)$
$\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow$
$Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp*],*safe*)
assume $\Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
then show $\Gamma\vdash\langle c\ Z,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$

by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal Z*)
assume c': *DynCom c* $\in$ *redexes* c'
have $\Gamma\vdash$ (*DynCom c*, *Normal Z*) $\rightarrow$ (*c Z*,*Normal Z*)
by (*rule step.DynCom*)
from *step-redexes* [*OF this c'*] **obtain** c'' **where**
step: $\Gamma\vdash$ (c', *Normal Z*) $\rightarrow$ (c'', *Normal Z*) **and** c'': *c Z* $\in$ *redexes* c''
by *blast*
note *steps* **also note** *step*
finally show $\exists\, c'$. $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^+$ (c', *Normal Z*) $\wedge$ *c Z* $\in$ *redexes* c'
using c'' **by** *blast*
next
fix *t*
assume $\Gamma\vdash\langle$*c Z*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*
thus $\Gamma\vdash\langle$*DynCom c*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*
by (*auto intro*: *exec.intros*)
next
fix *t*
assume $\Gamma\vdash\langle$*c Z*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*
thus $\Gamma\vdash\langle$*DynCom c*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*
by (*auto intro*: *exec.intros*)
qed
show *?case*
apply (*rule hoaret.DynCom*)
apply *safe*
apply (*rule hyp'*)
done
next
case (*Guard f g c*)
have *hyp-c*: $\forall\, Z$. $\Gamma,\Theta\vdash_{t/F}$
{*s*. *s*=*Z* $\wedge$ $\Gamma\vdash\langle$*c*,*Normal s*$\rangle$ $\Rightarrow\notin$({*Stuck*} $\cup$ *Fault* ‘ ($-F$)) $\wedge$
$\Gamma\vdash$*Call p*$\downarrow$*Normal* σ $\wedge$
($\exists\, c'$. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^+$ (c',*Normal s*) $\wedge$ *c* $\in$ *redexes* c')}
c
{*t*. $\Gamma\vdash\langle$*c*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*},
{*t*. $\Gamma\vdash\langle$*c*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*}
using *Guard.hyps* **by** *iprover*
show $\Gamma,\Theta\vdash_{t/F}$ {*s*. *s*=*Z* $\wedge$ $\Gamma\vdash\langle$*Guard f g c* ,*Normal s*$\rangle$ $\Rightarrow\notin$({*Stuck*} $\cup$ *Fault* ‘ ($-F$)) $\wedge$
$\Gamma\vdash$*Call p*$\downarrow$*Normal* σ $\wedge$
($\exists\, c'$. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^+$ (c',*Normal s*) $\wedge$ *Guard f g c* $\in$ *redexes* c')}
Guard f g c
{*t*. $\Gamma\vdash\langle$*Guard f g c* ,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*},
{*t*. $\Gamma\vdash\langle$*Guard f g c* ,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*}
proof (*cases f* $\in$ *F*)

case *True*
have $\Gamma,\Theta\vdash_{t/F}$ ($g \cap \{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \wedge$
$Guard\ f\ g\ c \in redexes\ c')\})$
c
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c*], *safe*)
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\ Z{\in}g$
thus $\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
assume c': $Guard\ f\ g\ c \in redexes\ c'$
assume $Z \in g$
from *this* **have** $\Gamma\vdash(Guard\ f\ g\ c,Normal\ Z) \rightarrow (c,Normal\ Z)$
by (*rule step.Guard*)
from *step-redexes* [*OF this c'*] **obtain** c'' **where**
step: $\Gamma\vdash\ (c',\ Normal\ Z) \rightarrow (c'',\ Normal\ Z)$ **and** c'': $c \in redexes\ c''$
by *blast*
note *steps* **also note** *step*
finally show $\exists\, c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z) \wedge c \in redexes$
c'
using c'' **by** *blast*
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\ Z \in g$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\ Z \in g$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
qed
from *True this* **show** *?thesis*
by (*rule conseqPre* [*OF Guarantee*]) *auto*
next
case *False*
have $\Gamma,\Theta\vdash_{t/F}$ ($g \cap \{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \wedge$
$Guard\ f\ g\ c \in redexes\ c')\})$

c
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c*], *safe*)
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
using *False*
by (*cases* $Z\in g$) (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
assume c': $Guard\ f\ g\ c \in redexes\ c'$

assume $Z \in g$
from *this* **have** $\Gamma\vdash(Guard\ f\ g\ c,Normal\ Z) \rightarrow (c,Normal\ Z)$
by (*rule step.Guard*)
from *step-redexes* [*OF this c'*] **obtain** c'' **where**
step: $\Gamma\vdash\ (c',\ Normal\ Z) \rightarrow (c'',\ Normal\ Z)$ **and** c'': $c \in redexes\ c''$
by *blast*
note *steps* **also note** *step*
finally show $\exists\, c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z) \wedge c \in redexes$
c'
using c'' **by** *blast*
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t$
using *False*
by (*cases* $Z\in g$) (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t$
using *False*
by (*cases* $Z\in g$) (*auto simp add*: *final-notin-def intro*: *exec.intros*)
qed
then show *?thesis*
apply (*rule conseqPre* [*OF hoaret.Guard*])
apply *clarify*
apply (*frule Guard-noFaultStuckD* [*OF - False*])
apply *auto*
done
qed
next
case *Throw*
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

$\Gamma\vdash Call\ p\downarrow Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,\ Normal\ \sigma)\ \rightarrow^{+}\ (c',Normal\ s)\ \wedge\ Throw\ \in\ redexes$
$c')\}$
$Throw$
$\{t.\ \Gamma\vdash\langle Throw,Normal\ Z\rangle\ \Rightarrow\ Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Throw,Normal\ Z\rangle\ \Rightarrow\ Abrupt\ t\}$
by (*rule conseqPre* [*OF hoaret.Throw*])
(*blast intro*: *exec.intros terminates.intros*)
next
case $(Catch\ c_1\ c_2)$
have *hyp-c1*:
$\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s=\ Z\ \wedge\ \Gamma\vdash\langle c_1,Normal\ s\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\ \text{`}\ (-F))\ \wedge$
$\Gamma\vdash Call\ p\ \downarrow\ Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma)\ \rightarrow^{+}\ (c',Normal\ s)\ \wedge$
$c_1\ \in\ redexes\ c')\}$
c_1
$\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle\ \Rightarrow\ Normal\ t\},\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle\ \Rightarrow\ Abrupt\ t\}$
using *Catch.hyps* **by** *iprover*
have *hyp-c2*:
$\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s=\ Z\ \wedge\ \Gamma\vdash\langle c_2,Normal\ s\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\ \text{`}\ (-F))\ \wedge$
$\Gamma\vdash Call\ p\downarrow\ Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma)\ \rightarrow^{+}\ (c',Normal\ s)\ \wedge\ c_2\ \in\ redexes\ c')\}$
c_2
$\{t.\ \Gamma\vdash\langle c_2,Normal\ Z\rangle\ \Rightarrow\ Normal\ t\},\{t.\ \Gamma\vdash\langle c_2,Normal\ Z\rangle\ \Rightarrow\ Abrupt\ t\}$
using *Catch.hyps* **by** *iprover*
have
$\Gamma,\Theta\vdash_{t/F}\ \{s.\ s\ =\ Z\ \wedge\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\ \text{`}\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p\downarrow\ Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma)\rightarrow^{+}(c',Normal\ s)\ \wedge$
$Catch\ c_1\ c_2\ \in\ redexes\ c')\}$
c_1
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle\ \Rightarrow\ Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle\ \Rightarrow\ Abrupt\ t\ \wedge$
$\Gamma\vdash\langle c_2,Normal\ t\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\text{`}(-F))\ \wedge\ \Gamma\vdash Call\ p\ \downarrow\ Normal\ \sigma$
$\wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma)\ \rightarrow^{+}\ (c',Normal\ t)\ \wedge\ c_2\ \in\ redexes\ c')\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*clarify*,*safe*)
assume $\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\ \text{`}\ (-F))$
thus $\Gamma\vdash\langle c_1,Normal\ Z\rangle\ \Rightarrow\notin(\{Stuck\}\ \cup\ Fault\ \text{`}\ (-F))$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma)\ \rightarrow^{+}\ (c',\ Normal\ Z)$
assume c': $Catch\ c_1\ c_2\ \in\ redexes\ c'$
from *steps redexes-subset* [*OF this*]
show $\exists\, c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma)\ \rightarrow^{+}\ (c',\ Normal\ Z)\ \wedge\ c_1\ \in\ redexes\ c'$
by (*auto iff*: *root-in-redexes*)
next

fix t
assume $\Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Normal\ t$
thus $\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*auto intro*: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t$
thus $\Gamma\vdash\langle c_2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c' t
assume *steps-c'*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z)$
assume *red*: $Catch\ c_1\ c_2 \in redexes\ c'$
assume *exec-c*$_1$: $\Gamma\vdash\ \langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t$
show $\exists\, c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ t) \wedge c_2 \in redexes\ c'$
proof −
note *steps-c'*
also
from *exec-impl-steps-Normal-Abrupt* [*OF exec-c*$_1$]
have $\Gamma\vdash\ (c_1,\ Normal\ Z) \rightarrow^* (Throw,\ Normal\ t)$**.**
from *steps-redexes-Catch* [*OF this red*]
obtain c'' **where**
steps-c'': $\Gamma\vdash\ (c',\ Normal\ Z) \rightarrow^* (c'',\ Normal\ t)$ **and**
Catch: $Catch\ Throw\ c_2 \in redexes\ c''$
by *blast*
note *steps-c''*
also
have *step-Catch*: $\Gamma\vdash\ (Catch\ Throw\ c_2,Normal\ t) \rightarrow (c_2,Normal\ t)$
by (*rule step.CatchThrow*)
from *step-redexes* [*OF step-Catch Catch*]
obtain c''' **where**
step-c''': $\Gamma\vdash\ (c'',\ Normal\ t) \rightarrow (c''',\ Normal\ t)$ **and**
c2: $c_2 \in redexes\ c'''$
by *blast*
note *step-c'''*
finally show *?thesis*
using *c2*
by *blast*
qed
qed
moreover
have $\Gamma,\Theta\vdash_{t/F}$ $\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t\ \wedge$
$\Gamma\vdash\langle c_2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\ \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ t) \wedge c_2 \in redexes\ c')\}$
c_2
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

by (*rule ConseqMGT* [*OF hyp-c2*]) (*fastforce intro*: *exec.intros*)
ultimately show *?case*
by (*rule hoaret.Catch*)
qed

To prove a procedure implementation correct it suffices to assume only the procedure specifications of procedures that actually occur during evaluation of the body.

lemma *Call-lemma*:
assumes *A*:
$\forall q \in dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ q\downarrow Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
assumes *pdef*: $p \in dom\ \Gamma$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$(\{\sigma\} \cap \{s.\ s=Z\ \wedge\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ s\})$
$the\ (\Gamma\ p)$
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule conseqPre*)
apply (*rule Call-lemma'* [*OF A*])
using *pdef*
apply (*fastforce intro*: *terminates.intros tranclp.r-into-trancl* [*of* (*step* Γ), *OF step.Call*] *root-in-redexes*)
done

lemma *Call-lemma-switch-Call-body*:
assumes
call: $\forall q \in dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ q\downarrow Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
assumes *p-defined*: $p \in dom\ \Gamma$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$(\{\sigma\} \cap \{s.\ s=Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ s\})$
$the\ (\Gamma\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*simp only*: *exec-Call-body'* [*OF p-defined*] *noFaultStuck-Call-body'* [*OF p-defined*]

terminates-Normal-Call-body [*OF p-defined*])
apply (*rule conseqPre*)
apply (*rule Call-lemma'*)
apply (*rule call*)
using *p-defined*
apply (*fastforce intro*: *terminates.intros tranclp.r-into-trancl* [*of* (*step* Γ), *OF step.Call*]
root-in-redexes)
done

lemma *MGT-Call*:
$\forall p \in dom\ \Gamma.\ \forall Z.$
  $\Gamma,\Theta \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
            $\Gamma\vdash(Call\ p)\downarrow Normal\ s\}$
           $(Call\ p)$
          $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
          $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*intro ballI allI*)
apply (*rule CallRec'* [**where** *Procs*=*dom* Γ **and**
    $P{=}\lambda p\ Z.\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                $\Gamma\vdash Call\ p\downarrow Normal\ s\}$ **and**
    $Q{=}\lambda p\ Z.\ \{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$ **and**
    $A{=}\lambda p\ Z.\ \{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$ **and**
    r=*termi-call-steps* Γ
    ])
apply *simp*
apply *simp*
apply (*rule wf-termi-call-steps*)
apply (*intro ballI allI*)
apply *simp*
apply (*rule Call-lemma-switch-Call-body* [*rule-format*, *simplified*])
apply (*rule hoaret.Asm*)
apply *fastforce*
apply *assumption*
done

lemma *CollInt-iff*: $\{s.\ P\ s\} \cap \{s.\ Q\ s\} = \{s.\ P\ s \wedge Q\ s\}$
  by *auto*

lemma *image-Un-conv*: $f\ `\ (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.\ \{x\ p\ Z\}) = (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.\ \{f\ (x\ p\ Z)\})$
  by (*auto iff*: *not-None-eq*)

Another proof of *MGT-Call*, maybe a little more readable

lemma
$\forall p \in dom\ \Gamma.\ \forall Z.$
  $\Gamma,\{\} \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
            $\Gamma\vdash(Call\ p)\downarrow Normal\ s\}$

$(Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

proof −
{
fix $p\ Z\ \sigma$
assume *defined*: $p \in dom\ \Gamma$
define *Specs* **where** $Specs = (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s{=}Z\ \wedge$
$\Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ s\}$,
p,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\})\})$
define *Specs-wf* **where** $Specs\text{-}wf\ p\ \sigma = (\lambda(P,q,Q,A).$
$(P \cap \{s.\ ((s,q),\sigma,p) \in termi\text{-}call\text{-}steps\ \Gamma\},\ q,\ Q,\ A))\ `\ Specs$ **for** $p\ \sigma$
have $\Gamma, Specs\text{-}wf\ p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle the\ (\Gamma\ p), Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ s\})$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p), Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p), Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule Call-lemma* [*rule-format*, *OF* - *defined*])
apply (*rule hoaret.Asm*)
apply (*clarsimp simp add*: *Specs-wf-def Specs-def image-Un-conv*)
apply (*rule-tac* $x{=}q$ **in** *bexI*)
apply (*rule-tac* $x{=}Z$ **in** *exI*)
apply (*clarsimp simp add*: *CollInt-iff*)
apply *auto*
done
hence $\Gamma, Specs\text{-}wf\ p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ s\})$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*simp only*: *exec-Call-body′* [*OF defined*]
noFaultStuck-Call-body′ [*OF defined*]
terminates-Normal-Call-body [*OF defined*])
} **note** *bdy=this*
show *?thesis*
apply (*intro ballI allI*)
apply (*rule hoaret.CallRec* [**where** $Specs{=}(\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s{=}Z\ \wedge$
$\Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ s\}$,

p,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}\})$,
OF - *wf-termi-call-steps* [*of* Γ] *refl*])
apply *fastforce*
apply *clarify*
apply (*rule conjI*)
apply *fastforce*
apply (*rule allI*)
apply (*simp* (*no-asm-use*) *only* : *Un-empty-left*)
apply (*rule bdy*)
apply *auto*
done
qed

theorem *hoaret-complete*: $\Gamma\models_{t/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\{\}\vdash_{t/F}\ P\ c\ Q,A$
by (*iprover intro*: *MGT-implies-complete MGT-lemma* [*OF MGT-Call*])

lemma *hoaret-complete'*:
assumes *cvalid*: $\Gamma,\Theta\models_{t/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
proof (*cases* $\Gamma\models_{t/F}\ P\ c\ Q,A$)
case *True*
hence $\Gamma,\{\}\vdash_{t/F}\ P\ c\ Q,A$
by (*rule hoaret-complete*)
thus $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
by (*rule hoaret-augment-context*) *simp*
next
case *False*
with *cvalid*
show *?thesis*
by (*rule ExFalso*)
qed

15.3 And Now: Some Useful Rules

15.3.1 Modify Return

lemma *ProcModifyReturn-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F}\ P\ call\ init\ p\ return'\ c\ Q,A$
assumes *valid-modif*:
$\forall\sigma.\ \Gamma,\Theta \models_{/UNIV} \{\sigma\}\ (Call\ p)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
assumes *res-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*:
$\forall s\ t.\ t \in ModifAbr\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$
shows $\Gamma,\Theta \models_{t/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$

proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  hence $\forall(P, p, Q, A)\in\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q,A$
    by (*auto simp add*: *validt-def*)
  then have *ctxt'*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma \models_{/UNIV} P\ (Call\ p)\ Q,A$
    by (*auto intro*: *valid-augment-Faults*)
  assume *exec*: $\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *exec*
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases rule*: *exec-call-Normal-elim*)
    fix *bdy t'*
    assume *bdy*: $\Gamma\ p = Some\ bdy$
    assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t'$
    assume *exec-c*: $\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle \Rightarrow t$
    from *exec-body bdy*
    have $\Gamma\vdash\langle(Call\ p\ ),Normal\ (init\ s)\rangle \Rightarrow Normal\ t'$
      by (*auto simp add*: *intro*: *exec.intros*)
    from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt' this*] *P*
    have $t' \in Modif\ (init\ s)$
      by *auto*
    with *res-modif* **have** $Normal\ (return'\ s\ t') = Normal\ (return\ s\ t')$
      by *simp*
    with *exec-body exec-c bdy*
    have $\Gamma\vdash\langle call\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
      by (*auto intro*: *exec-call*)
    from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
    show *?thesis*
      by *simp*
  next
    fix *bdy t'*
    assume *bdy*: $\Gamma\ p = Some\ bdy$
    assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Abrupt\ t'$
    assume *t*: $t = Abrupt\ (return\ s\ t')$
    also from *exec-body bdy*
    have $\Gamma\vdash\langle(Call\ p),Normal\ (init\ s)\rangle \Rightarrow Abrupt\ t'$
      by (*auto simp add*: *intro*: *exec.intros*)
    from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt' this*] *P*
    have $t' \in ModifAbr\ (init\ s)$
      by *auto*
    with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
      by *simp*
    finally have $t = Abrupt\ (return'\ s\ t')$ **.**
    with *exec-body bdy*
    have $\Gamma\vdash\langle call\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
      by (*auto intro*: *exec-callAbrupt*)
    from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*

show *?thesis*
by *simp*
next
fix *bdy f*
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Fault\ f$ **and**
t: $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-callFault*)
from *cvalidt-postD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy*
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Stuck$
$t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-callStuck*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
next
assume $\Gamma\ p = None\ t{=}Stuck$
hence $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-callUndefined*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
qed
next
fix *s*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q{,}A$
hence $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q{,}A$
by (*auto simp add*: *validt-def*)
then have *ctxt'*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/UNIV} P\ (Call\ p)\ Q{,}A$
by (*auto intro*: *valid-augment-Faults*)
assume *P*: $s \in P$
from *valid-call ctxt P*
have *call*: $\Gamma\vdash call\ init\ p\ return'\ c\!\downarrow Normal\ s$
by (*rule cvalidt-termD*)
show $\Gamma\vdash call\ init\ p\ return\ c \downarrow Normal\ s$
proof (*cases* $p \in dom\ \Gamma$)
case *True*
with *call* **obtain** *bdy* **where**
bdy: $\Gamma\ p = Some\ bdy$ **and** *termi-bdy*: $\Gamma\vdash bdy \downarrow Normal\ (init\ s)$ **and**
termi-c: $\forall t.\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow$
$\Gamma\vdash c\ s\ t \downarrow Normal\ (return'\ s\ t)$
by *cases auto*

{
fix t
assume *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
hence $\Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$
proof −
from *exec-bdy bdy*
have $\Gamma\vdash\langle(Call\ p\ ),Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt′ this*] *P*
res-modif
have $return'\ s\ t = return\ s\ t$
by *auto*
with *termi-c exec-bdy* **show** *?thesis* **by** *auto*
qed
}
with *bdy termi-bdy*
show *?thesis*
by (*iprover intro*: *terminates-call*)
next
case *False*
thus *?thesis*
by (*auto intro*: *terminates-callUndefined*)
qed
qed

lemma *ProcModifyReturn*:
assumes *spec*: $\Gamma,\Theta\vdash_{t/F}\ P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *res-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *ret-modifAbr*:
$\forall s\ t.\ t \in ModifAbr\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV}\ \{\sigma\}\ (Call\ p)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*rule hoaret-complete′*)
apply (*rule ProcModifyReturn-sound* [**where** *Modif=Modif* **and** *ModifAbr=ModifAbr*,

OF - - res-modif ret-modifAbr])
apply (*rule hoaret-sound* [*OF spec*])
using *modifies-spec*
apply (*blast intro*: *hoare-sound*)
done

lemma *ProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F} P\ call\ init\ p\ return'\ c\ Q,A$
assumes *valid-modif*:
$\forall \sigma.\ \Gamma,\Theta \models_{/F} \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
assumes *res-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$

assumes *ret-modifAbr*:
$\forall s\ t.\ t \in$ *ModifAbr* (*init s*) $\longrightarrow$ *return'* *s t* = *return s t*
shows $\Gamma,\Theta \models_{t/F}$ *P* (*call init p return c*) *Q*,*A*
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P$ (*Call p*) *Q*,*A*
hence *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{/F} P$ (*Call p*) *Q*,*A*
by (*auto simp add*: *validt-def*)
assume *exec*: $\Gamma\vdash\langle$*call init p return c*,*Normal s*$\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
from *exec*
show $t \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
proof (*cases rule*: *exec-call-Normal-elim*)
fix *bdy t'*
assume *bdy*: Γ *p* = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Normal t'*
assume *exec-c*: $\Gamma\vdash\langle$*c s t'*,*Normal* (*return s t'*)$\rangle \Rightarrow t$
from *exec-body bdy*
have $\Gamma\vdash\langle$(*Call p*) ,*Normal* (*init s*)$\rangle \Rightarrow$ *Normal t'*
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in$ *Modif* (*init s*)
by *auto*
with *res-modif* **have** *Normal* (*return' s t'*) = *Normal* (*return s t'*)
by *simp*
with *exec-body exec-c bdy*
have $\Gamma\vdash\langle$*call init p return' c*,*Normal s*$\rangle \Rightarrow t$
by (*auto intro*: *exec-call*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy t'*
assume *bdy*: Γ *p* = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Abrupt t'*
assume *t*: *t* = *Abrupt* (*return s t'*)
also
from *exec-body bdy*
have $\Gamma\vdash\langle$*Call p* ,*Normal* (*init s*)$\rangle \Rightarrow$ *Abrupt t'*
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in$ *ModifAbr* (*init s*)
by *auto*
with *ret-modifAbr* **have** *Abrupt* (*return s t'*) = *Abrupt* (*return' s t'*)
by *simp*
finally have *t* = *Abrupt* (*return' s t'*) **.**
with *exec-body bdy*
have $\Gamma\vdash\langle$*call init p return' c*,*Normal s*$\rangle \Rightarrow t$

by (*auto intro*: *exec-callAbrupt*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy f*
assume *bdy*: Γ *p* = *Some bdy*
assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Fault f* **and**
t: *t* = *Fault f*
with *bdy* **have** $\Gamma\vdash\langle$*call init p return' c* ,*Normal s*$\rangle \Rightarrow$ *t*
by (*auto intro*: *exec-callFault*)
from *cvalidt-postD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy*
assume *bdy*: Γ *p* = *Some bdy*
assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Stuck*
t = *Stuck*
with *bdy* **have** $\Gamma\vdash\langle$*call init p return' c*,*Normal s*$\rangle \Rightarrow$ *t*
by (*auto intro*: *exec-callStuck*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
next
assume Γ *p* = *None t*=*Stuck*
hence $\Gamma\vdash\langle$*call init p return' c*,*Normal s*$\rangle \Rightarrow$ *t*
by (*auto intro*: *exec-callUndefined*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
qed
next
fix *s*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
hence *ctxt'*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q,A$
by (*auto simp add*: *validt-def*)
assume *P*: $s \in P$
from *valid-call ctxt P*
have *call*: $\Gamma\vdash$*call init p return' c*$\downarrow$ *Normal s*
by (*rule cvalidt-termD*)
show $\Gamma\vdash$*call init p return c* $\downarrow$ *Normal s*
proof (*cases* $p \in$ *dom* Γ)
case *True*
with *call* **obtain** *bdy* **where**
bdy: Γ *p* = *Some bdy* **and** *termi-bdy*: $\Gamma\vdash$*bdy* $\downarrow$ *Normal* (*init s*) **and**
termi-c: $\forall t.$ $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Normal t* $\longrightarrow$
$\Gamma\vdash$*c s t* $\downarrow$ *Normal* (*return' s t*)
by *cases auto*

{
  fix t
  assume *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
  hence $\Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$
  proof −
    from *exec-bdy bdy*
    have $\Gamma\vdash\langle(Call\ p\ ),Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
      by (*auto simp add*: *intro*: *exec.intros*)
    from *cvalidD* [*OF valid-modif* [*rule-format*, *of init s*] *ctxt′ this*] *P*
      res-modif
    have *return′ s t* = *return s t*
      by *auto*
    with *termi-c exec-bdy* **show** *?thesis* **by** *auto*
  qed
}
with *bdy termi-bdy*
show *?thesis*
  by (*iprover intro*: *terminates-call*)
next
  case *False*
  thus *?thesis*
    by (*auto intro*: *terminates-callUndefined*)
qed
qed

lemma *ProcModifyReturnSameFaults*:
  assumes *spec*: $\Gamma,\Theta\vdash_{t/F}$ P (*call init p return′ c*) Q,A
  assumes *res-modif*:
  $\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
  assumes *ret-modifAbr*:
  $\forall s\ t.\ t \in ModifAbr\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
  assumes *modifies-spec*:
  $\forall\sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ (Call\ p)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  shows $\Gamma,\Theta\vdash_{t/F}$ P (*call init p return c*) Q,A
apply (*rule hoaret-complete′*)
apply (*rule ProcModifyReturnSameFaults-sound* [**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
        OF - - *res-modif ret-modifAbr*])
apply (*rule hoaret-sound* [*OF spec*])
using *modifies-spec*
apply (*blast intro*: *hoare-sound*)
done

15.3.2 DynCall

lemma *dynProcModifyReturn-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F}$ P *dynCall init p return′ c* Q,A
assumes *valid-modif*:
  $\forall s{\in}P.\ \forall\sigma.\ \Gamma,\Theta \models_{/UNIV} \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),(ModifAbr\ \sigma)$

assumes *ret-modif*:
$\forall s\ t.\ t \in \mathit{Modif}\ (\mathit{init}\ s) \longrightarrow \mathit{return}'\ s\ t = \mathit{return}\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in \mathit{ModifAbr}\ (\mathit{init}\ s) \longrightarrow \mathit{return}'\ s\ t = \mathit{return}\ s\ t$
shows $\Gamma,\Theta \models_{t/F} P\ (\mathit{dynCall}\ \mathit{init}\ p\ \mathit{return}\ c)\ Q,A$
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q,A$
hence $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{/F} P\ (\mathit{Call}\ p)\ Q,A$
by (*auto simp add*: *validt-def*)
then have *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{/UNIV} P\ (\mathit{Call}\ p)\ Q,A$
by (*auto intro*: *valid-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle \mathit{dynCall}\ \mathit{init}\ p\ \mathit{return}\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
assume *t-notin-F*: $t \notin \mathit{Fault}\ `\ F$
assume *P*: $s \in P$
with *valid-modif*
have *valid-modif'*:
$\forall \sigma.\ \Gamma,\Theta\models_{/UNIV} \{\sigma\}\ (\mathit{Call}\ (p\ s))\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
by *blast*
from *exec*
have $\Gamma\vdash\langle \mathit{call}\ \mathit{init}\ (p\ s)\ \mathit{return}\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
by (*cases rule*: *exec-dynCall-Normal-elim*)
then show $t \in \mathit{Normal}\ `\ Q \cup \mathit{Abrupt}\ `\ A$
proof (*cases rule*: *exec-call-Normal-elim*)
fix *bdy t'*
assume *bdy*: $\Gamma\ (p\ s) = \mathit{Some}\ \mathit{bdy}$
assume *exec-body*: $\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow \mathit{Normal}\ t'$
assume *exec-c*: $\Gamma\vdash\langle c\ s\ t',\mathit{Normal}\ (\mathit{return}\ s\ t')\rangle \Rightarrow t$
from *exec-body bdy*
have $\Gamma\vdash\langle \mathit{Call}\ (p\ s),\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow \mathit{Normal}\ t'$
by (*auto simp add*: *intro*: *exec.Call*)
from *cvalidD* [*OF valid-modif'* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in \mathit{Modif}\ (\mathit{init}\ s)$
by *auto*
with *ret-modif* **have** $\mathit{Normal}\ (\mathit{return}'\ s\ t') =$
$\mathit{Normal}\ (\mathit{return}\ s\ t')$
by *simp*
with *exec-body exec-c bdy*
have $\Gamma\vdash\langle \mathit{call}\ \mathit{init}\ (p\ s)\ \mathit{return}'\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-call*)
hence $\Gamma\vdash\langle \mathit{dynCall}\ \mathit{init}\ p\ \mathit{return}'\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy t'*
assume *bdy*: $\Gamma\ (p\ s) = \mathit{Some}\ \mathit{bdy}$
assume *exec-body*: $\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow \mathit{Abrupt}\ t'$

assume *t*: $t = Abrupt\ (return\ s\ t')$
also from *exec-body bdy*
have $\Gamma\vdash\langle Call\ (p\ s)\ ,Normal\ (init\ s)\rangle \Rightarrow Abrupt\ t'$
  by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif'* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in ModifAbr\ (init\ s)$
  by *auto*
with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
  by *simp*
finally have $t = Abrupt\ (return'\ s\ t')$ **.**
with *exec-body bdy*
have $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c,Normal\ s\rangle \Rightarrow t$
  by (*auto intro*: *exec-callAbrupt*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
  by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy f*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Fault\ f$ **and**
  t: $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
  by (*auto intro*: *exec-callFault*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
  by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
  by *blast*
next
fix *bdy*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Stuck$
  $t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
  by (*auto intro*: *exec-callStuck*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
  by (*rule exec-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cvalidt-postD*)
next
fix *bdy*
assume $\Gamma\ (p\ s) = None\ t{=}Stuck$
hence $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
  by (*auto intro*: *exec-callUndefined*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
  by (*rule exec-dynCall*)

from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
qed
next
fix *s*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q{,}A$
hence $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q{,}A$
by (*auto simp add*: *validt-def*)
then have *ctxt'*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/UNIV} P\ (Call\ p)\ Q{,}A$
by (*auto intro*: *valid-augment-Faults*)
assume *P*: $s \in P$
from *valid-call ctxt P*
have $\Gamma\vdash dynCall\ init\ p\ return'\ c{\downarrow}\ Normal\ s$
by (*rule cvalidt-termD*)
hence *call*: $\Gamma\vdash call\ init\ (p\ s)\ return'\ c{\downarrow}\ Normal\ s$
by *cases*
have $\Gamma\vdash call\ init\ (p\ s)\ return\ c \downarrow Normal\ s$
proof (*cases* $p\ s \in dom\ \Gamma$)
case *True*
with *call* **obtain** *bdy* **where**
bdy: $\Gamma\ (p\ s) = Some\ bdy$ **and** *termi-bdy*: $\Gamma\vdash bdy \downarrow Normal\ (init\ s)$ **and**
termi-c: $\forall t.\ \Gamma\vdash\langle bdy{,}Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow$
$\Gamma\vdash c\ s\ t \downarrow Normal\ (return'\ s\ t)$
by *cases auto*
{
fix *t*
assume *exec-bdy*: $\Gamma\vdash\langle bdy{,}Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
hence $\Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$
proof −
from *exec-bdy bdy*
have $\Gamma\vdash\langle Call\ (p\ s){,}Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif* [*rule-format*, *of s init s*] *ctxt' this*] *P*
ret-modif
have $return'\ s\ t = return\ s\ t$
by *auto*
with *termi-c exec-bdy* **show** *?thesis* **by** *auto*
qed
}
with *bdy termi-bdy*
show *?thesis*
by (*iprover intro*: *terminates-call*)
next
case *False*
thus *?thesis*
by (*auto intro*: *terminates-callUndefined*)
qed
thus $\Gamma\vdash dynCall\ init\ p\ return\ c \downarrow Normal\ s$

by (*iprover intro*: *terminates-dynCall*)
qed

lemma *dynProcModifyReturn*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{t/F}$ *P dynCall init p return′ c Q*,*A*
assumes *ret-modif*:
$\forall s\ t.\ t \in$ *Modif* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
assumes *modif*:
$\forall s \in P.\ \forall \sigma.$
$\Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta \vdash_{t/F}$ *P* (*dynCall init p return c*) *Q*,*A*
apply (*rule hoaret-complete′*)
apply (*rule dynProcModifyReturn-sound*
[**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
OF hoaret-sound [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-sound* [*OF modif* [*rule-format*]])
apply *assumption*
done

lemma *dynProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F}$ *P dynCall init p return′ c Q*,*A*
assumes *valid-modif*:
$\forall s{\in}P.\ \forall \sigma.\ \Gamma,\Theta \models_{/F} \{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
assumes *ret-modif*:
$\forall s\ t.\ t \in$ *Modif* (*init s*) $\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*) $\longrightarrow$ *return′ s t* = *return s t*
shows $\Gamma,\Theta \models_{t/F}$ *P* (*dynCall init p return c*) *Q*,*A*
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P$ (*Call p*) *Q*,*A*
hence *ctxt′*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F} P$ (*Call p*) *Q*,*A*
by (*auto simp add*: *validt-def*)
assume *exec*: $\Gamma\vdash\langle$*dynCall init p return c*,*Normal s*$\rangle \Rightarrow t$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
assume *P*: $s \in P$
with *valid-modif*
have *valid-modif′*:
$\forall \sigma.\ \Gamma,\Theta\models_{/F} \{\sigma\}$ (*Call* (*p s*)) (*Modif* σ),(*ModifAbr* σ)
by *blast*
from *exec*
have $\Gamma\vdash\langle$*call init* (*p s*) *return c*,*Normal s*$\rangle \Rightarrow t$
by (*cases rule*: *exec-dynCall-Normal-elim*)
then show $t \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
proof (*cases rule*: *exec-call-Normal-elim*)

fix *bdy* t'
assume *bdy*: Γ (*p s*) = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Normal* t'
assume *exec-c*: $\Gamma\vdash\langle$*c s* t',*Normal* (*return s* t')$\rangle \Rightarrow t$
from *exec-body bdy*
have $\Gamma\vdash\langle$*Call* (*p s*),*Normal* (*init s*)$\rangle \Rightarrow$ *Normal* t'
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif'* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in$ *Modif* (*init s*)
by *auto*
with *ret-modif* **have** *Normal* (*return' s* t') =
Normal (*return s* t')
by *simp*
with *exec-body exec-c bdy*
have $\Gamma\vdash\langle$*call init* (*p s*) *return' c*,*Normal s*$\rangle \Rightarrow t$
by (*auto intro*: *exec-call*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy* t'
assume *bdy*: Γ (*p s*) = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Abrupt* t'
assume *t*: *t* = *Abrupt* (*return s* t')
also from *exec-body bdy*
have $\Gamma\vdash\langle$*Call* (*p s*) ,*Normal* (*init s*)$\rangle \Rightarrow$ *Abrupt* t'
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif'* [*rule-format*, *of init s*] *ctxt' this*] *P*
have $t' \in$ *ModifAbr* (*init s*)
by *auto*
with *ret-modifAbr* **have** *Abrupt* (*return s* t') = *Abrupt* (*return' s* t')
by *simp*
finally have *t* = *Abrupt* (*return' s* t') **.**
with *exec-body bdy*
have $\Gamma\vdash\langle$*call init* (*p s*) *return' c*,*Normal s*$\rangle \Rightarrow t$
by (*auto intro*: *exec-callAbrupt*)
hence $\Gamma\vdash\langle$*dynCall init p return' c*,*Normal s*$\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy f*
assume *bdy*: Γ (*p s*) = *Some bdy*
assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle \Rightarrow$ *Fault f* **and**
t: *t* = *Fault f*
with *bdy* **have** $\Gamma\vdash\langle$*call init* (*p s*) *return' c* ,*Normal s*$\rangle \Rightarrow t$

by (*auto intro*: *exec-callFault*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
by *simp*
next
fix *bdy*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Stuck$
$t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-callStuck*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
next
fix *bdy*
assume $\Gamma\ (p\ s) = None\ t{=}Stuck$
hence $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle \Rightarrow t$
by (*auto intro*: *exec-callUndefined*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle \Rightarrow t$
by (*rule exec-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cvalidt-postD*)
qed
next
fix *s*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q{,}A$
hence *ctxt'*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q{,}A$
by (*auto simp add*: *validt-def*)
assume *P*: $s \in P$
from *valid-call ctxt P*
have $\Gamma\vdash dynCall\ init\ p\ return'\ c\downarrow\ Normal\ s$
by (*rule cvalidt-termD*)
hence *call*: $\Gamma\vdash call\ init\ (p\ s)\ return'\ c\downarrow\ Normal\ s$
by *cases*
have $\Gamma\vdash call\ init\ (p\ s)\ return\ c \downarrow Normal\ s$
proof (*cases* $p\ s \in dom\ \Gamma$)
case *True*
with *call* **obtain** *bdy* **where**
bdy: $\Gamma\ (p\ s) = Some\ bdy$ **and** *termi-bdy*: $\Gamma\vdash bdy \downarrow Normal\ (init\ s)$ **and**
termi-c: $\forall t.\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow$
$\Gamma\vdash c\ s\ t \downarrow Normal\ (return'\ s\ t)$
by *cases auto*
{

fix t
assume *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
hence $\Gamma\vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$
proof −
from *exec-bdy bdy*
have $\Gamma\vdash\langle Call\ (p\ s),Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
by (*auto simp add*: *intro*: *exec.intros*)
from *cvalidD* [*OF valid-modif* [*rule-format*, *of s init s*] *ctxt′ this*] *P*
ret-modif
have $return'\ s\ t = return\ s\ t$
by *auto*
with *termi-c exec-bdy* **show** *?thesis* **by** *auto*
qed
}
with *bdy termi-bdy*
show *?thesis*
by (*iprover intro*: *terminates-call*)
next
case *False*
thus *?thesis*
by (*auto intro*: *terminates-callUndefined*)
qed
thus $\Gamma\vdash dynCall\ init\ p\ return\ c \downarrow Normal\ s$
by (*iprover intro*: *terminates-dynCall*)
qed

lemma *dynProcModifyReturnSameFaults*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{t/F}\ P\ dynCall\ init\ p\ return'\ c\ Q,A$
assumes *ret-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif*:
$\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule hoaret-complete′*)
apply (*rule dynProcModifyReturnSameFaults-sound*
[**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
OF hoaret-sound [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-sound* [*OF modif* [*rule-format*]])
apply *assumption*
done

15.3.3 Conjunction of Postcondition

lemma *PostConjI-sound*:
assumes *valid-Q*: $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
assumes *valid-R*: $\Gamma,\Theta \models_{t/F} P\ c\ R,B$
shows $\Gamma,\Theta \models_{t/F} P\ c\ (Q \cap R),(A \cap B)$

proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *valid-Q ctxt exec P t-notin-F* **have** $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
    by (*rule cvalidt-postD*)
  moreover
  from *valid-R ctxt exec P t-notin-F* **have** $t \in Normal\ `\ R \cup Abrupt\ `\ B$
    by (*rule cvalidt-postD*)
  ultimately show $t \in Normal\ `\ (Q \cap R) \cup Abrupt\ `\ (A \cap B)$
    by *blast*
next
  fix *s*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  assume *P*: $s \in P$
  from *valid-Q ctxt P*
  show $\Gamma \vdash c \downarrow Normal\ s$
    by (*rule cvalidt-termD*)
qed

lemma *PostConjI*:
  assumes *deriv-Q*: $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
  assumes *deriv-R*: $\Gamma,\Theta \vdash_{t/F} P\ c\ R,B$
  shows $\Gamma,\Theta \vdash_{t/F} P\ c\ (Q \cap R),(A \cap B)$
apply (*rule hoaret-complete'*)
apply (*rule PostConjI-sound*)
apply (*rule hoaret-sound* [*OF deriv-Q*])
apply (*rule hoaret-sound* [*OF deriv-R*])
done

lemma *Merge-PostConj-sound*:
  assumes *validF*: $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
  assumes *validG*: $\Gamma,\Theta \models_{t/G} P'\ c\ R,X$
  assumes *F-G*: $F \subseteq G$
  assumes *P-P'*: $P \subseteq P'$
  shows $\Gamma,\Theta \models_{t/F} P\ c\ (Q \cap R),(A \cap X)$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
  with *F-G* **have** *ctxt'*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/G} P\ (Call\ p)\ Q,A$
    by (*auto intro*: *validt-augment-Faults*)
  assume *exec*: $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$
  assume *P*: $s \in P$
  with *P-P'* **have** *P'*: $s \in P'$
    by *auto*

assume *t-noFault*: $t \notin$ *Fault* ‘ F
show $t \in$ *Normal* ‘ $(Q \cap R) \cup$ *Abrupt* ‘ $(A \cap X)$
proof −
from *cvalidt-postD* [*OF validF* [*rule-format*] *ctxt exec P t-noFault*]
have $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A**.**
moreover from *this* **have** $t \notin$ *Fault* ‘ G
by *auto*
from *cvalidt-postD* [*OF validG* [*rule-format*] *ctxt′ exec P′ this*]
have $t \in$ *Normal* ‘ $R \cup$ *Abrupt* ‘ X **.**
ultimately show *?thesis* **by** *auto*
qed
next
fix s
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q{,}A$
assume P: $s \in P$
from *validF ctxt P*
show $\Gamma{\vdash}c \downarrow$ *Normal s*
by (*rule cvalidt-termD*)
qed

lemma *Merge-PostConj*:
assumes *validF*: $\Gamma{,}\Theta{\vdash}_{t/F}\ P\ c\ Q{,}A$
assumes *validG*: $\Gamma{,}\Theta{\vdash}_{t/G}\ P'\ c\ R{,}X$
assumes *F-G*: $F \subseteq G$
assumes *P-P′*: $P \subseteq P'$
shows $\Gamma{,}\Theta{\vdash}_{t/F}\ P\ c\ (Q \cap R){,}(A \cap X)$
apply (*rule hoaret-complete′*)
apply (*rule Merge-PostConj-sound* [*OF - - F-G P-P′*])
using *validF* **apply** (*blast intro*:*hoaret-sound*)
using *validG* **apply** (*blast intro*:*hoaret-sound*)
done

15.3.4 Guards and Guarantees

lemma *SplitGuards-sound*:
assumes *valid-c1*: $\Gamma{,}\Theta{\models}_{t/F}\ P\ c_1\ Q{,}A$
assumes *valid-c2*: $\Gamma{,}\Theta{\models}_{/F}\ P\ c_2\ UNIV{,}UNIV$
assumes c: $(c_1 \cap_g c_2) = Some\ c$
shows $\Gamma{,}\Theta{\models}_{t/F}\ P\ c\ Q{,}A$
proof (*rule cvalidtI*)
fix $s\ t$
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q{,}A$
hence *ctxt′*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F} P\ (Call\ p)\ Q{,}A$
by (*auto simp add*: *validt-def*)
assume *exec*: $\Gamma{\vdash}\langle c{,}Normal\ s\rangle \Rightarrow t$
assume P: $s \in P$

assume *t-notin-F*: $t \notin$ *Fault* ‘ F
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
proof (*cases t*)
  case *Normal*
  with *inter-guards-exec-noFault* [*OF c exec*]
  have $\Gamma\vdash\langle c_1,$*Normal s*$\rangle \Rightarrow t$ **by** *simp*
  from *valid-c1 ctxt this P t-notin-F*
  show *?thesis*
    by (*rule cvalidt-postD*)
next
  case *Abrupt*
  with *inter-guards-exec-noFault* [*OF c exec*]
  have $\Gamma\vdash\langle c_1,$*Normal s*$\rangle \Rightarrow t$ **by** *simp*
  from *valid-c1 ctxt this P t-notin-F*
  show *?thesis*
    by (*rule cvalidt-postD*)
next
  case (*Fault f*)
  assume *t*: t=*Fault f*
  with *exec inter-guards-exec-Fault* [*OF c*]
  have $\Gamma\vdash\langle c_1,$*Normal s*$\rangle \Rightarrow$ *Fault f* $\vee$ $\Gamma\vdash\langle c_2,$*Normal s*$\rangle \Rightarrow$ *Fault f*
    by *auto*
  then show *?thesis*
  proof (*cases rule*: *disjE* [*consumes 1*])
    assume $\Gamma\vdash\langle c_1,$*Normal s*$\rangle \Rightarrow$ *Fault f*
    from *cvalidt-postD* [*OF valid-c1 ctxt this P*] *t t-notin-F*
    show *?thesis*
      by *blast*
  next
    assume $\Gamma\vdash\langle c_2,$*Normal s*$\rangle \Rightarrow$ *Fault f*
    from *cvalidD* [*OF valid-c2 ctxt' this P*] *t t-notin-F*
    show *?thesis*
      by *blast*
  qed
next
  case *Stuck*
  with *inter-guards-exec-noFault* [*OF c exec*]
  have $\Gamma\vdash\langle c_1,$*Normal s*$\rangle \Rightarrow t$ **by** *simp*
  from *valid-c1 ctxt this P t-notin-F*
  show *?thesis*
    by (*rule cvalidt-postD*)
qed
next
fix *s*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P$ (*Call p*) Q,A
assume *P*: $s \in P$
show $\Gamma\vdash c \downarrow$ *Normal s*
proof −
  from *valid-c1 ctxt P*

have $\Gamma\vdash c_1 \downarrow$ *Normal s*
by (*rule cvalidt-termD*)
with *c* **show** *?thesis*
by (*rule inter-guards-terminates*)
qed
qed

lemma *SplitGuards*:
assumes *c*: $(c_1 \cap_g c_2) =$ *Some c*
assumes *deriv-c1*: $\Gamma,\Theta\vdash_{t/F} P\ c_1\ Q,A$
assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F} P\ c_2\ UNIV,UNIV$
shows $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule SplitGuards-sound* [*OF - - c*])
apply (*rule hoaret-sound* [*OF deriv-c1*])
apply (*rule hoare-sound* [*OF deriv-c2*])
done

lemma *CombineStrip-sound*:
assumes *valid*: $\Gamma,\Theta\models_{t/F} P\ c\ Q,A$
assumes *valid-strip*: $\Gamma,\Theta\models_{/\{\}} P$ (*strip-guards* $(-F)$ *c*) *UNIV*,*UNIV*
shows $\Gamma,\Theta\models_{t/\{\}} P\ c\ Q,A$
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/\{\}} P$ (*Call p*) Q,A
hence *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{/\{\}} P$ (*Call p*) Q,A
by (*auto simp add*: *validt-def*)
from *ctxt* **have** *ctxt''*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F} P$ (*Call p*) Q,A
by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ‘ {}
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ A
proof (*cases t*)
case (*Normal t'*)
from *cvalidt-postD* [*OF valid ctxt'' exec P*] *Normal*
show *?thesis*
by *auto*
next
case (*Abrupt t'*)
from *cvalidt-postD* [*OF valid ctxt'' exec P*] *Abrupt*
show *?thesis*
by *auto*
next
case (*Fault f*)
show *?thesis*
proof (*cases* $f \in F$)
case *True*

hence $f \notin -F$ **by** *simp*
with *exec Fault*
have $\Gamma\vdash\langle$*strip-guards* $(-F)$ c,*Normal s*$\rangle \Rightarrow$ *Fault f*
by (*auto intro*: *exec-to-exec-strip-guards-Fault*)
from *cvalidD* [*OF valid-strip ctxt′ this P*] *Fault*
have *False*
by *auto*
thus *?thesis* **..**
next
case *False*
with *cvalidt-postD* [*OF valid ctxt″ exec P*] *Fault*
show *?thesis*
by *auto*
qed
next
case *Stuck*
from *cvalidt-postD* [*OF valid ctxt″ exec P*] *Stuck*
show *?thesis*
by *auto*
qed
next
fix s
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/\{\}} P\ (Call\ p)\ Q{,}A$
hence *ctxt′*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}_{t/F}\ P\ (Call\ p)\ Q{,}A$
by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)
assume *P*: $s \in P$
show $\Gamma\vdash c \downarrow$ *Normal s*
proof $-$
from *valid ctxt′ P*
show $\Gamma\vdash c \downarrow$ *Normal s*
by (*rule cvalidt-termD*)
qed
qed

lemma *CombineStrip*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q{,}A$
assumes *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}\ P$ (*strip-guards* $(-F)$ c) *UNIV*,*UNIV*
shows $\Gamma,\Theta\vdash_{t/\{\}}\ P\ c\ Q{,}A$
apply (*rule hoaret-complete′*)
apply (*rule CombineStrip-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
apply (*iprover intro*: *hoare-sound* [*OF deriv-strip*])
done

lemma *GuardsFlip-sound*:
assumes *valid*: $\Gamma,\Theta{\models}_{t/F}\ P\ c\ Q{,}A$
assumes *validFlip*: $\Gamma,\Theta{\models}_{/-F}\ P\ c$ *UNIV*,*UNIV*
shows $\Gamma,\Theta{\models}_{t/\{\}}\ P\ c\ Q{,}A$

proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/\{\}}\ P\ (Call\ p)\ Q,A$
  from *ctxt* **have** *ctxt'*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)
  from *ctxt* **have** *ctxtFlip*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{/-F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)
  assume *exec*: $\Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin Fault\ `\ \{\}$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases t*)
    case (*Normal t'*)
    from *cvalidt-postD* [*OF valid ctxt' exec P*] *Normal*
    show *?thesis*
      by *auto*
  next
    case (*Abrupt t'*)
    from *cvalidt-postD* [*OF valid ctxt' exec P*] *Abrupt*
    show *?thesis*
      by *auto*
  next
    case (*Fault f*)
    show *?thesis*
    proof (*cases* $f \in F$)
      case *True*
      hence $f \notin -F$ **by** *simp*
      with *cvalidD* [*OF validFlip ctxtFlip exec P*] *Fault*
      have *False*
        by *auto*
      thus *?thesis* **..**
    next
      case *False*
      with *cvalidt-postD* [*OF valid ctxt' exec P*] *Fault*
      show *?thesis*
        by *auto*
    qed
  next
    case *Stuck*
    from *cvalidt-postD* [*OF valid ctxt' exec P*] *Stuck*
    show *?thesis*
      by *auto*
  qed
next
  fix *s*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/\{\}}\ P\ (Call\ p)\ Q,A$
  hence *ctxt'*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models_{t/F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *valid-augment-Faults simp add*: *validt-def*)

assume *P*: $s \in P$
show $\Gamma\vdash c \downarrow$ *Normal s*
proof −
from *valid ctxt' P*
show $\Gamma\vdash c \downarrow$ *Normal s*
by (*rule cvalidt-termD*)
qed
qed

lemma *GuardsFlip*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ *P c Q*,*A*
assumes *derivFlip*: $\Gamma,\Theta\vdash_{/-F}$ *P c UNIV*,*UNIV*
shows $\Gamma,\Theta\vdash_{t/\{\}}$ *P c Q*,*A*
apply (*rule hoaret-complete'*)
apply (*rule GuardsFlip-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
apply (*iprover intro*: *hoare-sound* [*OF derivFlip*])
done

lemma *MarkGuardsI-sound*:
assumes *valid*: $\Gamma,\Theta\models_{t/\{\}}$ *P c Q*,*A*
shows $\Gamma,\Theta\models_{t/\{\}}$ *P mark-guards f c Q*,*A*
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle$*mark-guards f c*,*Normal s*$\rangle \Rightarrow t$
from *exec-mark-guards-to-exec* [*OF exec*] **obtain** t' **where**
exec-c: $\Gamma\vdash\langle c$,*Normal s*$\rangle \Rightarrow t'$ **and**
t'*-noFault*: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ‘ {}
show $t \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
proof −
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-c P*]
have $t' \in$ *Normal* ‘ *Q* $\cup$ *Abrupt* ‘ *A*
by *blast*
with t'*-noFault*
show *?thesis*
by *auto*
qed
next
fix *s*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]
have $\Gamma\vdash c \downarrow$ *Normal s*.

thus $\Gamma\vdash$*mark-guards f c* $\downarrow$ *Normal s*
by (*rule terminates-to-terminates-mark-guards*)
qed

lemma *MarkGuardsI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/\{\}}$ *P c Q*,*A*
shows $\Gamma,\Theta\vdash_{t/\{\}}$ *P mark-guards f c Q*,*A*
apply (*rule hoaret-complete'*)
apply (*rule MarkGuardsI-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *MarkGuardsD-sound*:
assumes *valid*: $\Gamma,\Theta\models_{t/\{\}}$ *P mark-guards f c Q*,*A*
shows $\Gamma,\Theta\models_{t/\{\}}$ *P c Q*,*A*
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ‘ {}
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ *A*
proof (*cases isFault t*)
case *True*
with *exec-to-exec-mark-guards-Fault exec*
obtain f' **where** $\Gamma\vdash\langle$*mark-guards f c*,*Normal s*$\rangle \Rightarrow$ *Fault* f'
by (*fastforce elim*: *isFaultE*)
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt this P*]
have *False*
by *auto*
thus *?thesis* **..**
next
case *False*
from *exec-to-exec-mark-guards* [*OF exec False*]
obtain f' **where** $\Gamma\vdash\langle$*mark-guards f c*,*Normal s*$\rangle \Rightarrow t$
by *auto*
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt this P*]
show *?thesis*
by *auto*
qed
next
fix *s*
assume *ctxt*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]
have $\Gamma\vdash$*mark-guards f c* $\downarrow$ *Normal s*.
thus $\Gamma\vdash c \downarrow$ *Normal s*

  by (*rule terminates-mark-guards-to-terminates*)
qed

lemma *MarkGuardsD*:
 assumes *deriv*: $\Gamma,\Theta\vdash_{t/\{\}}$ *P mark-guards f c Q*,*A*
 shows $\Gamma,\Theta\vdash_{t/\{\}}$ *P c Q*,*A*
apply (*rule hoaret-complete'*)
apply (*rule MarkGuardsD-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *MergeGuardsI-sound*:
 assumes *valid*: $\Gamma,\Theta\models_{t/F}$ *P c Q*,*A*
 shows $\Gamma,\Theta\models_{t/F}$ *P merge-guards c Q*,*A*
proof (*rule cvalidtI*)
 fix *s t*
 assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}$ *P* (*Call p*) *Q*,*A*
 assume *exec-merge*: $\Gamma\vdash\langle$*merge-guards c*,*Normal s*$\rangle \Rightarrow t$
 from *exec-merge-guards-to-exec* [*OF exec-merge*]
 have *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow t$ **.**
 assume *P*: $s \in P$
 assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
 from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec P t-notin-F*]
 show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ *A*.
next
 fix *s*
 assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}$ *P* (*Call p*) *Q*,*A*
 assume *P*: $s \in P$
 from *cvalidt-termD* [*OF valid ctxt P*]
 have $\Gamma\vdash c \downarrow$ *Normal s*.
 thus $\Gamma\vdash$*merge-guards c* $\downarrow$ *Normal s*
  by (*rule terminates-to-terminates-merge-guards*)
qed

lemma *MergeGuardsI*:
 assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ *P c Q*,*A*
 shows $\Gamma,\Theta\vdash_{t/F}$ *P merge-guards c Q*,*A*
apply (*rule hoaret-complete'*)
apply (*rule MergeGuardsI-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *MergeGuardsD-sound*:
 assumes *valid*: $\Gamma,\Theta\models_{t/F}$ *P merge-guards c Q*,*A*
 shows $\Gamma,\Theta\models_{t/F}$ *P c Q*,*A*
proof (*rule cvalidtI*)
 fix *s t*
 assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}$ *P* (*Call p*) *Q*,*A*

assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
from *exec-to-exec-merge-guards* [*OF exec*]
have *exec-merge*: $\Gamma\vdash\langle merge\text{-}guards\ c,Normal\ s\rangle \Rightarrow t$.
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault$ ` F
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-merge P t-notin-F*]
show $t \in Normal$ ` $Q \cup Abrupt$ ` A.
next
fix *s*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]
have $\Gamma\vdash merge\text{-}guards\ c \downarrow Normal\ s$.
thus $\Gamma\vdash c \downarrow Normal\ s$
by (*rule terminates-merge-guards-to-terminates*)
qed

lemma *MergeGuardsD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/F} P\ merge\text{-}guards\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule MergeGuardsD-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *SubsetGuards-sound*:
assumes *c-c'*: $c \subseteq_g c'$
assumes *valid*: $\Gamma,\Theta\models_{t/\{\}} P\ c'\ Q,A$
shows $\Gamma,\Theta\models_{t/\{\}} P\ c\ Q,A$
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}} P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
from *exec-to-exec-subseteq-guards* [*OF c-c' exec*] **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c',Normal\ s\rangle \Rightarrow t'$ **and**
t'-noFault: $\neg\ isFault\ t' \longrightarrow t' = t$
by *blast*
assume *P*: $s \in P$
assume *t-noFault*: $t \notin Fault$ ` $\{\}$
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-c' P*] *t'-noFault t-noFault*
show $t \in Normal$ ` $Q \cup Abrupt$ ` A
by *auto*
next
fix *s*
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/\{\}} P\ (Call\ p)\ Q,A$
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]

have *termi-c'*: $\Gamma\vdash c' \downarrow$ *Normal s*.
from *cvalidt-postD* [*OF valid ctxt - P*]
have *noFault-c'*: $\Gamma\vdash\langle c',$*Normal s*$\rangle \Rightarrow\notin$*Fault* ` *UNIV*
by (*auto simp add*: *final-notin-def*)
from *termi-c' c-c' noFault-c'*
show $\Gamma\vdash c \downarrow$ *Normal s*
by (*rule terminates-fewer-guards*)
qed

lemma *SubsetGuards*:
assumes *c-c'*: $c \subseteq_g c'$
assumes *deriv*: $\Gamma,\Theta\vdash_{t/\{\}}$ $P\ c'\ Q,A$
shows $\Gamma,\Theta\vdash_{t/\{\}}$ $P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule SubsetGuards-sound* [*OF c-c'*])
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *NormalizeD-sound*:
assumes *valid*: $\Gamma,\Theta\models_{t/F}$ *P* (*normalize c*) *Q*,*A*
shows $\Gamma,\Theta\models_{t/F}$ $P\ c\ Q,A$
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}$ *P* (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow t$
hence *exec-norm*: $\Gamma\vdash\langle$*normalize c*,*Normal s*$\rangle \Rightarrow t$
by (*rule exec-to-exec-normalize*)
assume *P*: $s \in P$
assume *noFault*: $t \notin$ *Fault* ` *F*
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-norm P noFault*]
show $t \in$ *Normal* ` $Q \cup$ *Abrupt* ` *A*.
next
fix *s*
assume *ctxt*: $\forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}$ *P* (*Call p*) *Q*,*A*
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]
have $\Gamma\vdash$*normalize c* $\downarrow$ *Normal s*.
thus $\Gamma\vdash c \downarrow$ *Normal s*
by (*rule terminates-normalize-to-terminates*)
qed

lemma *NormalizeD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ *P* (*normalize c*) *Q*,*A*
shows $\Gamma,\Theta\vdash_{t/F}$ $P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule NormalizeD-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *NormalizeI-sound*:
assumes *valid*: $\Gamma,\Theta\models_{t/F}$ P c Q,A
shows $\Gamma,\Theta\models_{t/F}$ P (*normalize c*) Q,A
proof (*rule cvalidtI*)
fix s t
assume *ctxt*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
assume $\Gamma\vdash\langle normalize\ c,Normal\ s\rangle \Rightarrow t$
hence *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
by (*rule exec-normalize-to-exec*)
assume *P*: $s \in P$
assume *noFault*: $t \notin Fault$ ` F
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec P noFault*]
show $t \in Normal$ ` $Q \cup Abrupt$ ` A.
next
fix s
assume *ctxt*: $\forall(P, p, Q, A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
assume *P*: $s \in P$
from *cvalidt-termD* [*OF valid ctxt P*]
have $\Gamma\vdash c \downarrow Normal\ s$.
thus $\Gamma\vdash normalize\ c \downarrow Normal\ s$
by (*rule terminates-to-terminates-normalize*)
qed

lemma *NormalizeI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ P c Q,A
shows $\Gamma,\Theta\vdash_{t/F}$ P (*normalize c*) Q,A
apply (*rule hoaret-complete'*)
apply (*rule NormalizeI-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

15.3.5 Restricting the Procedure Environment

lemma *validt-restrict-to-validt*:
assumes *validt-c*: $\Gamma|_M\models_{t/F}$ P c Q,A
shows $\Gamma\models_{t/F}$ P c Q,A
proof −
from *validt-c*
have *valid-c*: $\Gamma|_M\models_{/F}$ P c Q,A **by** (*simp add*: *validt-def*)
hence $\Gamma\models_{/F}$ P c Q,A **by** (*rule valid-restrict-to-valid*)
moreover
{
fix s
assume *P*: $s \in P$
have $\Gamma\vdash c\downarrow Normal\ s$
proof −
from *P validt-c* **have** $\Gamma|_M\vdash c\downarrow Normal\ s$

by (*auto simp add*: *validt-def*)
moreover
from *P valid-c*
have $\Gamma|_M\vdash\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *valid-def final-notin-def*)
ultimately show *?thesis*
by (*rule terminates-restrict-to-terminates*)
qed
}
ultimately show *?thesis*
by (*auto simp add*: *validt-def*)
qed

lemma *augment-procs*:
assumes *deriv-c*: $\Gamma|_M,\{\}\vdash_{t/F}\ P\ c\ Q,A$
shows $\Gamma,\{\}\vdash_{t/F}\ P\ c\ Q,A$
apply (*rule hoaret-complete*)
apply (*rule validt-restrict-to-validt*)
apply (*insert hoaret-sound* [*OF deriv-c*])
by (*simp add*: *cvalidt-def*)

15.3.6 Miscellaneous

lemma *augment-Faults*:
assumes *deriv-c*: $\Gamma,\{\}\vdash_{t/F}\ P\ c\ Q,A$
assumes *F*: $F \subseteq F'$
shows $\Gamma,\{\}\vdash_{t/F'}\ P\ c\ Q,A$
apply (*rule hoaret-complete*)
apply (*rule validt-augment-Faults* [*OF - F*])
apply (*insert hoaret-sound* [*OF deriv-c*])
by (*simp add*: *cvalidt-def*)

lemma *TerminationPartial-sound*:
assumes *termination*: $\forall s \in P.\ \Gamma\vdash c\downarrow Normal\ s$
assumes *partial-corr*: $\Gamma,\Theta\models_{/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\models_{t/F}\ P\ c\ Q,A$
using *termination partial-corr*
by (*auto simp add*: *cvalidt-def validt-def cvalid-def*)

lemma *TerminationPartial*:
assumes *partial-deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
assumes *termination*: $\forall s \in P.\ \Gamma\vdash c\downarrow Normal\ s$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule TerminationPartial-sound* [*OF termination*])
apply (*rule hoare-sound* [*OF partial-deriv*])
done

lemma *TerminationPartialStrip*:
  assumes *partial-deriv*: $\Gamma,\Theta\vdash_{/F}$ P c Q,A
  assumes *termination*: $\forall\, s \in P.$ *strip* F' $\Gamma\vdash$*strip-guards* F' $c\downarrow$*Normal* s
  shows $\Gamma,\Theta\vdash_{t/F}$ P c Q,A
proof −
  from *termination* **have** $\forall\, s \in P.$ $\Gamma\vdash c\downarrow$*Normal* s
    by (*auto intro*: *terminates-strip-guards-to-terminates*
      terminates-strip-to-terminates)
  with *partial-deriv*
  show *?thesis*
    by (*rule TerminationPartial*)
qed

lemma *SplitTotalPartial*:
  assumes *termi*: $\Gamma,\Theta\vdash_{t/F}$ P c Q',A'
  assumes *part*: $\Gamma,\Theta\vdash_{/F}$ P c Q,A
  shows $\Gamma,\Theta\vdash_{t/F}$ P c Q,A
proof −
  from *hoaret-sound* [*OF termi*] *hoare-sound* [*OF part*]
  have $\Gamma,\Theta\models_{t/F}$ P c Q,A
    by (*fastforce simp add*: *cvalidt-def validt-def cvalid-def valid-def*)
  thus *?thesis*
    by (*rule hoaret-complete′*)
qed

lemma *SplitTotalPartial′*:
  assumes *termi*: $\Gamma,\Theta\vdash_{t/UNIV}$ P c Q',A'
  assumes *part*: $\Gamma,\Theta\vdash_{/F}$ P c Q,A
  shows $\Gamma,\Theta\vdash_{t/F}$ P c Q,A
proof −
  from *hoaret-sound* [*OF termi*] *hoare-sound* [*OF part*]
  have $\Gamma,\Theta\models_{t/F}$ P c Q,A
    by (*fastforce simp add*: *cvalidt-def validt-def cvalid-def valid-def*)
  thus *?thesis*
    by (*rule hoaret-complete′*)
qed

end

16 Derived Hoare Rules for Total Correctness

theory *HoareTotal* **imports** *HoareTotalProps* **begin**

lemma *conseq-no-aux*:
  $[\![\Gamma,\Theta \vdash_{t/F} P'\ c\ Q',A';$
    $\forall\, s.\ s \in P \longrightarrow (s{\in}P' \wedge (Q' \subseteq Q) \wedge (A' \subseteq A))]\!]$

$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
by (*rule conseq* [**where** P'=$\lambda Z.\ P'$ **and** Q'=$\lambda Z.\ Q'$ **and** A'=$\lambda Z.\ A'$]) *auto*

If for example a specification for a "procedure pointer" parameter is in the precondition we can extract it with this rule

lemma *conseq-exploit-pre*:
$[\![\forall s \in P.\ \Gamma,\Theta \vdash_{t/F} (\{s\} \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
apply (*rule Conseq*)
apply *clarify*
apply (*rule-tac* x=$\{s\} \cap P$ **in** *exI*)
apply (*rule-tac* x=Q **in** *exI*)
apply (*rule-tac* x=A **in** *exI*)
by *simp*

lemma *conseq*:$[\![\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\forall s.\ s \in P \longrightarrow (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,A$
by (*rule Conseq'*) *blast*

lemma *Lem*:$[\![\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A)\}]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ (lem\ x\ c)\ Q,A$
apply (*unfold lem-def*)
apply (*erule conseq*)
apply *blast*
done

lemma *LemAnno*:
assumes *conseq*: $P \subseteq \{s.\ \exists Z.\ s{\in}P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow t \in Q) \wedge (\forall t.\ t \in A'\ Z \longrightarrow t \in A)\}$
assumes *lem*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (lem\ x\ c)\ Q,A$
apply (*rule Lem* [*OF lem*])
using *conseq*
by *blast*

lemma *LemAnnoNoAbrupt*:
assumes *conseq*: $P \subseteq\ \{s.\ \exists Z.\ s{\in}P'\ Z \wedge (\forall t.\ t \in Q'\ Z \longrightarrow t \in Q)\}$
assumes *lem*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (lem\ x\ c)\ Q,\{\}$

apply (*rule Lem* [*OF lem*])
using *conseq*
by *blast*

lemma *TrivPost*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
$\Longrightarrow$
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ \mathit{UNIV},\mathit{UNIV}$
apply (*rule allI*)
apply (*erule conseq*)
apply *auto*
done

lemma *TrivPostNoAbr*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),\{\}$
$\Longrightarrow$
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ \mathit{UNIV},\{\}$
apply (*rule allI*)
apply (*erule conseq*)
apply *auto*
done

lemma *DynComConseq*:
assumes $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \ \Gamma,\Theta\vdash_{t/F} P'\ (c\ s)\ Q',A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
shows $\Gamma,\Theta\vdash_{t/F} P$ *DynCom* $c\ Q,A$
using *assms*
apply –
apply (*rule hoaret.DynCom*)
apply *clarsimp*
apply (*rule hoaret.Conseq*)
apply *clarsimp*
apply *blast*
done

lemma *SpecAnno*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))\}$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z)$
assumes *bdy-constant*: $\forall Z.\ c\ Z = c\ \mathit{undefined}$
shows $\Gamma,\Theta\vdash_{t/F} P$ (*specAnno* $P'\ c\ Q'\ A'$) Q,A
proof –
from *spec bdy-constant*
have $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ \mathit{undefined})\ (Q'\ Z),(A'\ Z)$
apply –
apply (*rule allI*)
apply (*erule-tac x=Z* **in** *allE*)
apply (*erule-tac x=Z* **in** *allE*)
apply *simp*
done
with *consequence* **show** *?thesis*

apply (*simp add*: *specAnno-def*)
apply (*erule conseq*)
apply *blast*
done
qed

lemma *SpecAnno'*:
$[\![P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q) \wedge (\forall t.\ t \in A'\ Z \longrightarrow t \in\ A)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z);$
$\forall Z.\ c\ Z = c\ undefined$
$]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (specAnno\ P'\ c\ Q'\ A')\ Q,A$
apply (*simp only*: *subset-iff* [*THEN sym*])
apply (*erule* (*1*) *SpecAnno*)
apply *assumption*
done

lemma *SpecAnnoNoAbrupt*:
$[\![P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),\{\};$
$\forall Z.\ c\ Z = c\ undefined$
$]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (specAnno\ P'\ c\ Q'\ (\lambda s.\ \{\}))\ Q,A$
apply (*rule SpecAnno'*)
apply *auto*
done

lemma *Skip*: $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ Skip\ Q,A$
by (*rule hoaret.Skip* [*THEN conseqPre*],*simp*)

lemma *Basic*: $P \subseteq \{s.\ (f\ s) \in Q\} \Longrightarrow\ \Gamma,\Theta\vdash_{t/F} P\ (Basic\ f)\ Q,A$
by (*rule hoaret.Basic* [*THEN conseqPre*])

lemma *BasicCond*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s{\in}Q) \wedge (\neg\ b\ s \longrightarrow g\ s{\in}Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ Basic\ (\lambda s.\ if\ b\ s\ then\ f\ s\ else\ g\ s)\ Q,A$
apply (*rule Basic*)
apply *auto*
done

lemma *Spec*: $P \subseteq \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (Spec\ r)\ Q,A$
by (*rule hoaret.Spec* [*THEN conseqPre*])

lemma *SpecIf*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ \mathit{Spec}\ (\mathit{if\text{-}rel}\ b\ f\ g\ h)\ Q,A$
apply (*rule Spec*)
apply (*auto simp add*: *if-rel-def*)
done

lemma *Seq* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{t/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ \mathit{Seq}\ c_1\ c_2\ Q,A$
by (*rule hoaret.Seq*)

lemma *SeqSwap*:
$[\![\Gamma,\Theta\vdash_{t/F}\ R\ c2\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ P\ c1\ R,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ \mathit{Seq}\ c1\ c2\ Q,A$
by (*rule Seq*)

lemma *BSeq*:
$[\![\Gamma,\Theta\vdash_{t/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{bseq}\ c_1\ c_2)\ Q,A$
by (*unfold bseq-def*) (*rule Seq*)

lemma *Cond*:
assumes *wp*: $P \subseteq \{s.\ (s\in b \longrightarrow s\in P_1) \wedge (s\notin b \longrightarrow s\in P_2)\}$
assumes *deriv-c1*: $\Gamma,\Theta\vdash_{t/F}\ P_1\ c_1\ Q,A$
assumes *deriv-c2*: $\Gamma,\Theta\vdash_{t/F}\ P_2\ c_2\ Q,A$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Cond}\ b\ c_1\ c_2)\ Q,A$
proof (*rule hoaret.Cond* [*THEN conseqPre*])
from *deriv-c1*
show $\Gamma,\Theta\vdash_{t/F}\ (\{s.\ (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap b)\ c_1\ Q,A$
by (*rule conseqPre*) *blast*
next
from *deriv-c2*
show $\Gamma,\Theta\vdash_{t/F}\ (\{s.\ (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap -\ b)\ c_2\ Q,A$
by (*rule conseqPre*) *blast*
qed (*insert wp*)

lemma *CondSwap*:
$[\![\Gamma,\Theta\vdash_{t/F}\ P1\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ P2\ c2\ Q,A;$
$P \subseteq \{s.\ (s\in b \longrightarrow s\in P1) \wedge (s\notin b \longrightarrow s\in P2)\}]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Cond}\ b\ c1\ c2)\ Q,A$
by (*rule Cond*)

lemma *Cond'*:
$[\![P \subseteq \{s.\ (b \subseteq P1) \wedge (-\ b \subseteq P2)\};\Gamma,\Theta\vdash_{t/F}\ P1\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ P2\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Cond}\ b\ c1\ c2)\ Q,A$
by (*rule CondSwap*) *blast+*

lemma *CondInv*:
  assumes *wp*: $P \subseteq Q$
  assumes *inv*: $Q \subseteq \{s.\ (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
  assumes *deriv-c1*: $\Gamma,\Theta\vdash_{t/F} P_1\ c_1\ Q,A$
  assumes *deriv-c2*: $\Gamma,\Theta\vdash_{t/F} P_2\ c_2\ Q,A$
  shows $\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$
proof −
  from *wp inv*
  have $P \subseteq \{s.\ (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
    by *blast*
  from *Cond* [*OF this deriv-c1 deriv-c2*]
  show *?thesis* **.**
qed

lemma *CondInv'*:
  assumes *wp*: $P \subseteq I$
  assumes *inv*: $I \subseteq \{s.\ (s{\in}b \longrightarrow s{\in}P_1) \wedge (s{\notin}b \longrightarrow s{\in}P_2)\}$
  assumes *wp'*: $I \subseteq Q$
  assumes *deriv-c1*: $\Gamma,\Theta\vdash_{t/F} P_1\ c_1\ I,A$
  assumes *deriv-c2*: $\Gamma,\Theta\vdash_{t/F} P_2\ c_2\ I,A$
  shows $\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$
proof −
  from *CondInv* [*OF wp inv deriv-c1 deriv-c2*]
  have $\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c_1\ c_2)\ I,A$ **.**
  from *conseqPost* [*OF this wp' subset-refl*]
  show *?thesis* **.**
qed

lemma *switchNil*:
  $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (switch\ v\ [])\ Q,A$
  by (*simp add*: *Skip*)

lemma *switchCons*:
  $[\![P \subseteq \{s.\ (v\ s \in V \longrightarrow s \in P_1) \wedge (v\ s \notin V \longrightarrow s \in P_2)\};$
        $\Gamma,\Theta\vdash_{t/F} P_1\ c\ Q,A;$
        $\Gamma,\Theta\vdash_{t/F} P_2\ (switch\ v\ vs)\ Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (switch\ v\ ((V,c)\#vs))\ Q,A$
  by (*simp add*: *Cond*)

lemma *Guard*:
$[\![P \subseteq g \cap R;\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ Guard\ f\ g\ c\ Q,A$
apply (*rule HoareTotalDef.Guard* [*THEN conseqPre*, *of - - - - R*])
apply (*erule conseqPre*)

apply *auto*
done

lemma *GuardSwap*:
$\llbracket\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ P \subseteq g \cap R\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ \mathit{Guard}\ f\ g\ c\ Q,A$
by (*rule Guard*)

lemma *Guarantee*:
$\llbracket P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*, *of* - - - - - $\{s.\ s \in g \longrightarrow s \in R\}$])
apply *assumption*
apply (*erule conseqPre*)
apply *auto*
done

lemma *GuaranteeSwap*:
$\llbracket\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
by (*rule Guarantee*)

lemma *GuardStrip*:
$\llbracket P \subseteq R;\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*])
apply *auto*
done

lemma *GuardStripSwap*:
$\llbracket\Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ P \subseteq R;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
by (*rule GuardStrip*)

lemma *GuaranteeStrip*:
$\llbracket P \subseteq R;\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{guaranteeStrip}\ f\ g\ c)\ Q,A$
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeStripSwap*:
$\llbracket\Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ P \subseteq R;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{guaranteeStrip}\ f\ g\ c)\ Q,A$
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeAsGuard*:
$\llbracket P \subseteq g \cap R;\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A\rrbracket$

$\Longrightarrow \Gamma,\Theta \vdash_{t/F}$ *P guaranteeStrip f g c Q*,*A*
by (*unfold guaranteeStrip-def*) (*rule Guard*)

lemma *GuaranteeAsGuardSwap*:
$[\![\ \Gamma,\Theta \vdash_{t/F} R\ c\ Q,A;\ P \subseteq g \cap R]\!]$
$\Longrightarrow \Gamma,\Theta \vdash_{t/F}$ *P guaranteeStrip f g c Q*,*A*
by (*rule GuaranteeAsGuard*)

lemma *GuardsNil*:
$\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A \Longrightarrow$
$\Gamma,\Theta \vdash_{t/F} P$ (*guards* [] *c*) *Q*,*A*
by *simp*

lemma *GuardsCons*:
$\Gamma,\Theta \vdash_{t/F}$ *P Guard f g* (*guards gs c*) *Q*,*A* $\Longrightarrow$
$\Gamma,\Theta \vdash_{t/F}$ *P* (*guards* ((*f*,*g*)#*gs*) *c*) *Q*,*A*
by *simp*

lemma *GuardsConsGuaranteeStrip*:
$\Gamma,\Theta \vdash_{t/F}$ *P guaranteeStrip f g* (*guards gs c*) *Q*,*A* $\Longrightarrow$
$\Gamma,\Theta \vdash_{t/F}$ *P* (*guards* (*guaranteeStripPair f g*#*gs*) *c*) *Q*,*A*
by (*simp add*: *guaranteeStripPair-def guaranteeStrip-def*)

lemma *While*:
assumes *P-I*: $P \subseteq I$
assumes *deriv-body*:
$\forall \sigma.\ \Gamma,\Theta \vdash_{t/F} (\{\sigma\} \cap I \cap b)\ c\ (\{t.\ (t,\ \sigma) \in V\} \cap I),A$
assumes *I-Q*: $I \cap -b \subseteq Q$
assumes *wf*: *wf V*
shows $\Gamma,\Theta \vdash_{t/F}$ *P* (*whileAnno b I V c*) *Q*,*A*
proof −
from *wf deriv-body P-I I-Q*
show *?thesis*
apply (*unfold whileAnno-def*)
apply (*erule conseqPrePost* [*OF HoareTotalDef*.*While*])
apply *auto*
done
qed

lemma *WhileInvPost*:
assumes *P-I*: $P \subseteq I$
assumes *termi-body*:
$\forall \sigma.\ \Gamma,\Theta \vdash_{t/UNIV} (\{\sigma\} \cap I \cap b)\ c\ (\{t.\ (t,\ \sigma) \in V\} \cap P),A$
assumes *deriv-body*:
$\Gamma,\Theta \vdash_{/F} (I \cap b)\ c\ I,A$

assumes *I-Q*: $I \cap -b \subseteq Q$
assumes *wf*: *wf V*
shows $\Gamma,\Theta\vdash_{t/F}$ P (*whileAnno* b I V c) Q,A
proof −
have $\forall\sigma.$ $\Gamma,\Theta\vdash_{t/F}$ ($\{\sigma\} \cap I \cap b$) c ($\{t.$ $(t, \sigma) \in V\} \cap I$),$A$
proof
fix σ
from *hoare-sound* [*OF deriv-body*] *hoaret-sound* [*OF termi-body* [*rule-format*, *of* σ]]
have $\Gamma,\Theta\models_{t/F}$ ($\{\sigma\} \cap I \cap b$) c ($\{t.$ $(t, \sigma) \in V\} \cap I$),$A$
by (*fastforce simp add*: *cvalidt-def validt-def cvalid-def valid-def*)
then
show $\Gamma,\Theta\vdash_{t/F}$ ($\{\sigma\} \cap I \cap b$) c ($\{t.$ $(t, \sigma) \in V\} \cap I$),$A$
by (*rule hoaret-complete′*)
qed

from *While* [*OF P-I this I-Q wf*]
show *?thesis* **.**
qed

lemma $\Gamma,\Theta\vdash_{/F}$ ($P \cap b$) c $Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}$ ($P \cap b$) (*Seq* c (*Guard* f Q *Skip*)) Q,A
oops

J will be instantiated by tactic with $gs' \cap I$ for those guards that are not stripped.

lemma *WhileAnnoG*:
$\Gamma,\Theta\vdash_{t/F}$ P (*guards gs*
(*whileAnno* b J V (*Seq* c (*guards gs Skip*)))) Q,A
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F}$ P (*whileAnnoG gs* b I V c) Q,A
by (*simp add*: *whileAnnoG-def whileAnno-def while-def*)

This form stems from *strip-guards* F (*whileAnnoG gs* b I V c)

lemma *WhileNoGuard′*:
assumes *P-I*: $P \subseteq I$
assumes *deriv-body*: $\forall\sigma.$ $\Gamma,\Theta\vdash_{t/F}$ ($\{\sigma\} \cap I \cap b$) c ($\{t.$ $(t, \sigma) \in V\} \cap I$),$A$
assumes *I-Q*: $I \cap -b \subseteq Q$
assumes *wf*: *wf V*
shows $\Gamma,\Theta\vdash_{t/F}$ P (*whileAnno* b I V (*Seq* c *Skip*)) Q,A
apply (*rule While* [*OF P-I - I-Q wf*])
apply (*rule allI*)
apply (*rule Seq*)
apply (*rule deriv-body* [*rule-format*])
apply (*rule hoaret.Skip*)
done

lemma *WhileAnnoFix*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}I\ Z \wedge (I\ Z \cap -b \subseteq Q))\ \}$
assumes *bdy*: $\forall\,Z\ \sigma.\ \Gamma,\Theta\vdash_{t/F}$ $(\{\sigma\} \cap I\ Z \cap b)$ $(c\ Z)$ $(\{t.\ (t,\ \sigma) \in V\ Z\} \cap I\ Z),A$
assumes *bdy-constant*: $\forall\,Z.\ c\ Z = c$ *undefined*
assumes *wf*: $\forall\,Z.$ *wf* $(V\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}$ P (*whileAnnoFix b I V c*) Q,A
proof −
  from *bdy bdy-constant*
  have *bdy'*: $\bigwedge Z.\ \forall\,\sigma.\ \Gamma,\Theta\vdash_{t/F}$ $(\{\sigma\} \cap I\ Z \cap b)$ (*c undefined*)
      $(\{t.\ (t,\ \sigma) \in V\ Z\} \cap I\ Z),A$
    apply −
    apply (*erule-tac x=Z* **in** *allE*)
    apply (*erule-tac x=Z* **in** *allE*)
    apply *simp*
    done
  have $\forall\,Z.\ \Gamma,\Theta\vdash_{t/F}$ $(I\ Z)$ (*whileAnnoFix b I V c*) $(I\ Z \cap -b),A$
    apply *rule*
    apply (*unfold whileAnnoFix-def*)
    apply (*rule hoaret.While*)
    apply (*rule wf* [*rule-format*])
    apply (*rule bdy'*)
    done
  then
  show *?thesis*
    apply (*rule conseq*)
    using *consequence*
    by *blast*
qed

lemma *WhileAnnoFix'*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}I\ Z \wedge$
      $(\forall\,t.\ t \in I\ Z \cap -b \longrightarrow t \in Q))\ \}$
assumes *bdy*: $\forall\,Z\ \sigma.\ \Gamma,\Theta\vdash_{t/F}$ $(\{\sigma\} \cap I\ Z \cap b)$ $(c\ Z)$ $(\{t.\ (t,\ \sigma) \in V\ Z\} \cap I\ Z),A$
assumes *bdy-constant*: $\forall\,Z.\ c\ Z = c$ *undefined*
assumes *wf*: $\forall\,Z.$ *wf* $(V\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}$ P (*whileAnnoFix b I V c*) Q,A
  apply (*rule WhileAnnoFix* [*OF - bdy bdy-constant wf*])
  using *consequence* **by** *blast*

lemma *WhileAnnoGFix*:
assumes *whileAnnoFix*:
  $\Gamma,\Theta\vdash_{t/F}$ P (*guards gs*
      (*whileAnnoFix b J V* ($\lambda Z.$ (*Seq* (*c Z*) (*guards gs Skip*))))) Q,A
shows $\Gamma,\Theta\vdash_{t/F}$ P (*whileAnnoGFix gs b I V c*) Q,A
  using *whileAnnoFix*
  by (*simp add*: *whileAnnoGFix-def whileAnnoFix-def while-def*)

lemma *Bind*:

assumes *adapt*: $P \subseteq \{s.\ s \in P'\ s\}$
assumes *c*: $\forall\, s.\ \Gamma,\Theta \vdash_{t/F} (P'\ s)\ (c\ (e\ s))\ Q,A$
shows $\Gamma,\Theta \vdash_{t/F} P\ (bind\ e\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge s \in P'\ Z\}$ **and** $Q'=\lambda Z.\ Q$ **and** $A'=\lambda Z.\ A$])
apply (*rule allI*)
apply (*unfold bind-def*)
apply (*rule HoareTotalDef.DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule conseqPre*)
apply (*rule c* [*rule-format*])
apply *blast*
using *adapt*
apply *blast*
done

lemma *Block*:
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall\, s.\ \Gamma,\Theta \vdash_{t/F} (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall\, s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
shows $\Gamma,\Theta \vdash_{t/F} P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge init\ s \in P'\ Z\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule HoareTotalDef.DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in R\ Z\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ Z\ t \in R\ Z\ t\}$ **and**
$Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule HoareTotalDef.DynCom*)
apply (*clarsimp*)
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in A\}$ **in** *HoareTotalDef.Catch*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*

apply *simp*
apply (*rule bdy* [*rule-format*])
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *BlockSwap*:
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (block\ init\ bdy\ return\ c)\ Q,A$
  using *adapt bdy c*
  by (*rule Block*)

lemma *BlockSpec*:
  assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
    $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
    $(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
  assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
  assumes *bdy*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ Z)\ bdy\ (Q'\ Z),(A'\ Z)$
  shows $\Gamma,\Theta\vdash_{t/F}\ P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ init\ s \in P'\ Z\ \wedge$
    $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
    $(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule HoareTotalDef.DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ s\ t \in R\ s\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ s\ t \in R\ s\ t\}$ **and**
    $Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule HoareTotalDef.DynCom*)
apply (*clarsimp*)
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ s\ t \in A\}$ **in** *HoareTotalDef.Catch*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\}$ **in** *Seq*)

apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule conseq* [*OF bdy*])
apply *clarsimp*
apply *blast*
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *Throw*: $P \subseteq A \Longrightarrow \Gamma,\Theta\vdash_{t/F} P$ *Throw* Q,A
by (*rule hoaret.Throw* [*THEN conseqPre*])

lemmas *Catch* = *hoaret.Catch*
lemma *CatchSwap*: $[\![\Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ P\ c_1\ Q,R]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P$ *Catch* $c_1\ c_2\ Q,A$
by (*rule hoaret.Catch*)

lemma *raise*: $P \subseteq \{s.\ f\ s \in A\} \Longrightarrow \Gamma,\Theta\vdash_{t/F} P$ *raise* $f\ Q,A$
apply (*simp add*: *raise-def*)
apply (*rule Seq*)
apply (*rule Basic*)
apply (*assumption*)
apply (*rule Throw*)
apply (*rule subset-refl*)
done

lemma *condCatch*: $[\![\Gamma,\Theta\vdash_{t/F}\ P\ c_1\ Q,((b \cap R) \cup (-b \cap A));\Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A]\!]$
$\Longrightarrow\ \Gamma,\Theta\vdash_{t/F}\ P$ *condCatch* $c_1\ b\ c_2\ Q,A$
apply (*simp add*: *condCatch-def*)
apply (*rule Catch*)
apply *assumption*
apply (*rule CondSwap*)
apply (*assumption*)
apply (*rule hoaret.Throw*)
apply *blast*
done

lemma *condCatchSwap*: $[\![\Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A;\ \Gamma,\Theta\vdash_{t/F}\ P\ c_1\ Q,((b \cap R) \cup (-b \cap A))]\!]$
$\Longrightarrow\ \Gamma,\Theta\vdash_{t/F}\ P$ *condCatch* $c_1\ b\ c_2\ Q,A$
by (*rule condCatch*)

lemma *ProcSpec*:

assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow \mathit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ Z \longrightarrow \mathit{return}\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ \mathit{Call}\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F} P\ (\mathit{call}\ \mathit{init}\ p\ \mathit{return}\ c)\ Q,A$
using *adapt c p*
apply (*unfold call-def*)
by (*rule BlockSpec*)

lemma *ProcSpec'*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ Z \wedge$
$(\forall t \in Q'\ Z.\ \mathit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ Z.\ \mathit{return}\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ \mathit{Call}\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F} P\ (\mathit{call}\ \mathit{init}\ p\ \mathit{return}\ c)\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
apply (*insert adapt*)
apply *clarsimp*
apply (*drule* (*1*) *subsetD*)
apply (*clarsimp*)
apply (*rule-tac x=Z* **in** *exI*)
apply *blast*
done

lemma *ProcSpecNoAbrupt*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow \mathit{return}\ s\ t \in R\ s\ t)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ \mathit{Call}\ p\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{t/F} P\ (\mathit{call}\ \mathit{init}\ p\ \mathit{return}\ c)\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
using *adapt*
apply *simp*
done

lemma *FCall*:
$\Gamma,\Theta\vdash_{t/F} P\ (\mathit{call}\ \mathit{init}\ p\ \mathit{return}\ (\lambda s\ t.\ c\ (\mathit{result}\ t)))\ Q,A$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (\mathit{fcall}\ \mathit{init}\ p\ \mathit{return}\ \mathit{result}\ c)\ Q,A$
by (*simp add*: *fcall-def*)

lemma *ProcRec*:
assumes *deriv-bodies*:
$\forall p{\in}\mathit{Procs}.$
$\forall \sigma\ Z.\ \Gamma,\Theta\cup(\bigcup q{\in}\mathit{Procs}.\ \bigcup Z.$

$\{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\})$
$\vdash_{t/F}$ $(\{\sigma\} \cap P\ p\ Z)$ $(the\ (\Gamma\ p))$ $(Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *Procs-defined*: *Procs* $\subseteq$ *dom* Γ
shows $\forall\, p{\in}Procs.\ \forall\, Z.$
$\Gamma,\Theta\vdash_{t/F}(P\ p\ Z)$ *Call p* $(Q\ p\ Z),(A\ p\ Z)$
by (*intro strip*)
(*rule HoareTotalDef.CallRec'*
[*OF - Procs-defined wf deriv-bodies*],
simp-all)

lemma *ProcRec'*:
assumes *ctxt*:
$\Theta'{=}(\lambda\sigma\ p.\ \Theta\cup(\bigcup q{\in}Procs.$
$\bigcup Z.\ \{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\}))$
assumes *deriv-bodies*:
$\forall\, p{\in}Procs.$
$\forall\,\sigma\ Z.\ \Gamma,\Theta'\ \sigma\ p\vdash_{t/F}$ $(\{\sigma\} \cap P\ p\ Z)$ $(the\ (\Gamma\ p))$ $(Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *Procs-defined*: *Procs* $\subseteq$ *dom* Γ
shows $\forall\, p{\in}Procs.\ \forall\, Z.$ $\Gamma,\Theta\vdash_{t/F}(P\ p\ Z)$ *Call p* $(Q\ p\ Z),(A\ p\ Z)$
using *ctxt deriv-bodies*
apply *simp*
apply (*erule ProcRec* [*OF - wf Procs-defined*])
done

lemma *ProcRecList*:
assumes *deriv-bodies*:
$\forall\, p{\in}set\ Procs.$
$\forall\,\sigma\ Z.\ \Gamma,\Theta\cup(\bigcup q{\in}set\ Procs.\ \bigcup Z.$
$\{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\})$
$\vdash_{t/F}$ $(\{\sigma\} \cap P\ p\ Z)$ $(the\ (\Gamma\ p))$ $(Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *dist*: *distinct Procs*
assumes *Procs-defined*: *set Procs* $\subseteq$ *dom* Γ
shows $\forall\, p{\in}set\ Procs.\ \forall\, Z.$
$\Gamma,\Theta\vdash_{t/F}(P\ p\ Z)$ *Call p* $(Q\ p\ Z),(A\ p\ Z)$
using *deriv-bodies wf Procs-defined*
by (*rule ProcRec*)

lemma *ProcRecSpecs*:
$[\![\forall\,\sigma.\ \forall\,(P,p,Q,A) \in Specs.$
$\Gamma,\Theta\cup\ ((\lambda(P,q,Q,A).\ (P \cap \{s.\ ((s,q),(\sigma,p)) \in r\},q,Q,A))$ ` *Specs*)
$\vdash_{t/F}$ $(\{\sigma\} \cap P)$ $(the\ (\Gamma\ p))$ $Q,A;$
wf r;
$\forall\,(P,p,Q,A) \in Specs.\ p \in dom\ \Gamma]\!]$
$\Longrightarrow \forall\,(P,p,Q,A) \in Specs.\ \Gamma,\Theta\vdash_{t/F}\ P\ (Call\ p)\ Q,A$

apply (*rule ballI*)
apply (*case-tac x*)
apply (*rename-tac x P p Q A*)
apply *simp*
apply (*rule hoaret.CallRec*)
apply *auto*
done

lemma *ProcRec1*:
 assumes *deriv-body*:
 $\forall \sigma\ Z.\ \Gamma,\Theta \cup (\bigcup Z.\ \{(P\ Z \cap \{s.\ ((s,p),\ \sigma,p) \in r\},p,Q\ Z,A\ Z)\})$
 $\vdash_{t/F} (\{\sigma\} \cap P\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
 assumes *wf*: *wf r*
 assumes *p-defined*: $p \in \mathit{dom}\ \Gamma$
 shows $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P\ Z)\ \mathit{Call}\ p\ (Q\ Z),(A\ Z)$
proof −
 from *deriv-body wf p-defined*
 have $\forall p\in\{p\}.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F} (P\ Z)\ \mathit{Call}\ p\ (Q\ Z),(A\ Z)$
  apply −
  apply (*rule ProcRec* [**where** $A=\lambda p.\ A$ **and** $P=\lambda p.\ P$ **and** $Q=\lambda p.\ Q$])
  apply *simp-all*
  done
 thus *?thesis*
  by *simp*
qed

lemma *ProcNoRec1*:
 assumes *deriv-body*:
 $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
 assumes *p-defined*: $p \in \mathit{dom}\ \Gamma$
 shows $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P\ Z)\ \mathit{Call}\ p\ (Q\ Z),(A\ Z)$
proof −
 have $\forall \sigma\ Z.\ \Gamma,\Theta\vdash_{t/F} (\{\sigma\} \cap P\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
  by (*blast intro*: *conseqPre deriv-body* [*rule-format*])
 with *p-defined* **have** $\forall \sigma\ Z.\ \Gamma,\Theta \cup (\bigcup Z.\ \{(P\ Z \cap \{s.\ ((s,p),\ \sigma,p) \in \{\}\},$
 $p,Q\ Z,A\ Z)\})$
 $\vdash_{t/F} (\{\sigma\} \cap P\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ Z),(A\ Z)$
  by (*blast intro*: *hoaret-augment-context*)
 from *this*
 show *?thesis*
  by (*rule ProcRec1*) (*auto simp add*: *p-defined*)
qed

lemma *ProcBody*:
 assumes *WP*: $P \subseteq P'$
 assumes *deriv-body*: $\Gamma,\Theta\vdash_{t/F} P'\ \mathit{body}\ Q,A$
 assumes *body*: $\Gamma\ p = \mathit{Some}\ \mathit{body}$
 shows $\Gamma,\Theta\vdash_{t/F} P\ \mathit{Call}\ p\ Q,A$

apply (*rule conseqPre* [*OF* - *WP*])
apply (*rule ProcNoRec1* [*rule-format*, **where** $P=\lambda Z.\ P'$ **and** $Q=\lambda Z.\ Q$ **and** $A=\lambda Z.\ A$])
apply (*insert body*)
apply *simp*
apply (*rule hoaret-augment-context* [*OF deriv-body*])
apply *blast*
apply *fastforce*
done

lemma *CallBody*:
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{t/F} (P'\ s)\ body\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *body*: $\Gamma\ p = Some\ body$
shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*unfold call-def*)
apply (*rule Block* [*OF adapt* - *c*])
apply (*rule allI*)
apply (*rule ProcBody* [**where** $\Gamma=\Gamma$, *OF* - *bdy* [*rule-format*] *body*])
apply *simp*
done

lemmas *ProcModifyReturn* = *HoareTotalProps.ProcModifyReturn*
lemmas *ProcModifyReturnSameFaults* = *HoareTotalProps.ProcModifyReturnSameFaults*

lemma *ProcModifyReturnNoAbr*:
assumes *spec*: $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturn* [*OF spec result-conform* - *modifies-spec*]) *simp*

lemma *ProcModifyReturnNoAbrSameFaults*:
assumes *spec*: $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturnSameFaults* [*OF spec result-conform* - *modifies-spec*]) *simp*

lemma *DynProc*:

assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ s\ Z \wedge$
$(\forall t.\ t \in Q'\ s\ Z \longrightarrow\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ s\ Z \longrightarrow return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s\in\ P.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ s\ Z)\ Call\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ dynCall\ init\ p\ return\ c\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge s \in P\}$
and $Q'=\lambda Z.\ Q$ **and** $A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold dynCall-def call-def block-def*)
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])
apply *clarsimp*
apply (*rename-tac* Z')
apply (*rule-tac* $R=Q'\ Z\ Z'$ **in** *Seq*)
apply (*rule CatchSwap*)
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule subset-refl*)
apply (*rule Basic*)
apply (*rule subset-refl*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\ Z'\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule-tac* $Q'=Q'\ Z\ Z'$ **and** $A'=A'\ Z\ Z'$ **in** *conseqPost*)
using *p*
apply *clarsimp*
apply *simp*
apply *clarsimp*
apply (*rule-tac* $P'=\lambda Z''.\ \{t.\ t=Z'' \wedge return\ Z\ t \in R\ Z\ t\}$ **and**
$Q'=\lambda Z''.\ Q$ **and** $A'=\lambda Z''.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
done

lemma *DynProc'*:

assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ s\ Z\ \wedge$
$(\forall t \in Q'\ s\ Z.\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ s\ Z.\ return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s\in P.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ s\ Z)\ Call\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta\vdash_{t/F} P\ dynCall\ init\ p\ return\ c\ Q,A$
proof −
from *adapt* **have** $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ s\ Z\ \wedge$
$(\forall t.\ t \in Q'\ s\ Z \longrightarrow\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ s\ Z \longrightarrow return\ s\ t \in A)\}$
by *blast*
from *this c p* **show** *?thesis*
by (*rule DynProc*)
qed

lemma *DynProcStaticSpec*:
assumes *adapt*: $P \subseteq \{s.\ s \in S \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall s\in S.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ Call\ (p\ s)\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof −
from *adapt* **have** *P-S*: $P \subseteq S$
by *blast*
have $\Gamma,\Theta\vdash_{t/F} (P \cap S)\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProc* [**where** $P'=\lambda s\ Z.\ P'\ Z$ **and** $Q'=\lambda s\ Z.\ Q'\ Z$
and $A'=\lambda s\ Z.\ A'\ Z$, *OF - c*])
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])
apply *clarsimp*
using *spec*
apply *clarsimp*
done
thus *?thesis*
by (*rule conseqPre*) (*insert P-S*,*blast*)
qed

lemma *DynProcProcPar*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ Call\ q\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProcStaticSpec* [**where** $S=\{s.\ p\ s = q\}$,*simplified*, *OF adapt c*])

using *spec*
apply *simp*
done

lemma *DynProcProcParNoAbrupt*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ Call\ q\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof −
  have $P \subseteq \{s.\ p\ s = q \wedge (\exists\ Z.\ init\ s \in P'\ Z \wedge$
  $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
  $(\forall t.\ t \in \{\} \longrightarrow return\ s\ t \in A))\}$
    (**is** $P \subseteq ?P'$)
  proof
    fix *s*
    assume *P*: $s \in P$
    with *adapt* **obtain** *Z* **where**
      Pre: $p\ s = q \wedge init\ s \in P'\ Z$ **and**
      adapt-Norm: $\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau$
      by *blast*
    from *adapt-Norm*
    have $\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t$
      by *auto*
    then
    show $s \in ?P'$
      using *Pre* **by** *blast*
  qed
  note $P = this$
  show *?thesis*
    apply −
    apply (*rule DynProcStaticSpec* [**where** $S=\{s.\ p\ s = q\}$,*simplified*, *OF P c*])
    apply (*insert spec*)
    apply *auto*
    done
qed

lemma *DynProcModifyReturnNoAbr*:
  assumes *to-prove*: $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return'\ c)\ Q,A$
  assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  assumes *modif-clause*:
  $\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta \vdash_{/UNIV} \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),\{\}$
  shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof −
  from *ret-nrm-modif*

have $\forall s\ t.\ t\ \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *iprover*
then
have *ret-nrm-modif'*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *simp*
have *ret-abr-modif'*: $\forall s\ t.\ t \in \{\}$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *simp*
from *to-prove ret-nrm-modif' ret-abr-modif' modif-clause* **show** *?thesis*
by (*rule dynProcModifyReturn*)
qed

lemma *ProcDynModifyReturnNoAbrSameFaults*:
assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}\ P\ (dynCall\ init\ p\ return'\ c)\ Q,A$
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),\{\}$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof $-$
from *ret-nrm-modif*
have $\forall s\ t.\ t\ \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *iprover*
then
have *ret-nrm-modif'*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *simp*
have *ret-abr-modif'*: $\forall s\ t.\ t \in \{\}$
$\longrightarrow return'\ s\ t = return\ s\ t$
by *simp*
from *to-prove ret-nrm-modif' ret-abr-modif' modif-clause* **show** *?thesis*
by (*rule dynProcModifyReturnSameFaults*)
qed

lemma *ProcProcParModifyReturn*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcPar* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}\ P'\ (dynCall\ init\ p\ return'\ c)\ Q,A$
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-abr-modif*: $\forall s\ t.\ t \in (ModifAbr\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV}\ \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),(ModifAbr\ \sigma)$

shows $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return c*) Q,A
proof –
from *to-prove* **have** $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
ret-abr-modif
have $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
by (*rule dynProcModifyReturn*) (*insert modif-clause,auto*)
from *this q* **show** *?thesis*
by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnSameFaults*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcPar* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-abr-modif*: $\forall s\ t.\ t \in (ModifAbr\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall\sigma.\ \Gamma,\Theta\vdash_{/F}$ $\{\sigma\}$ *Call q* (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return c*) Q,A
proof –
from *to-prove*
have $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
ret-abr-modif
have $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
by (*rule dynProcModifyReturnSameFaults*) (*insert modif-clause,auto*)
from *this q* **show** *?thesis*
by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnNoAbr*:
assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}$ P' (*dynCall init p return' c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif-clause*:
$\forall\sigma.\ \Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ (*Call q*) (*Modif* σ),{}
shows $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return c*) Q,A

proof −
  from *to-prove* **have** $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
    by (*rule conseqPre*) *blast*
  from *this ret-nrm-modif*
  have $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
    by (*rule DynProcModifyReturnNoAbr*) (*insert modif-clause*,*auto*)
  from *this q* **show** *?thesis*
    by (*rule conseqPre*)
qed

lemma *ProcProcParModifyReturnNoAbrSameFaults*:
  assumes *q*: $P \subseteq \{s.\ p\ s = q\} \cap P'$
    — *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in P', so the vcg can simplify it.
  assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}$ P' (*dynCall init p return' c*) Q,A
  assumes *ret-nrm-modif*: $\forall s\ t.\ t \in$ (*Modif* (*init s*))
                    $\longrightarrow$ *return' s t* = *return s t*
  assumes *modif-clause*:
      $\forall \sigma.\ \Gamma,\Theta\vdash_{/F}$ $\{\sigma\}$ (*Call q*) (*Modif* σ),$\{\}$
  shows $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return c*) Q,A
proof −
  from *to-prove* **have**
    $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return' c*) Q,A
    by (*rule conseqPre*) *blast*
  from *this ret-nrm-modif*
  have $\Gamma,\Theta\vdash_{t/F}$ ($\{s.\ p\ s = q\} \cap P'$) (*dynCall init p return c*) Q,A
    by (*rule ProcDynModifyReturnNoAbrSameFaults*) (*insert modif-clause*,*auto*)
  from *this q* **show** *?thesis*
    by (*rule conseqPre*)
qed

lemma *MergeGuards-iff*: $\Gamma,\Theta\vdash_{t/F}$ P *merge-guards c* Q,A = $\Gamma,\Theta\vdash_{t/F}$ P c Q,A
  by (*auto intro*: *MergeGuardsI MergeGuardsD*)

lemma *CombineStrip'*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}$ P c' Q,A
  assumes *deriv-strip-triv*: $\Gamma,\{\}\vdash_{/\{\}}$ P c'' *UNIV*,*UNIV*
  assumes c'': c''= *mark-guards False* (*strip-guards* ($-F$) c')
  assumes *c*: *merge-guards c* = *merge-guards* (*mark-guards False* c')
  shows $\Gamma,\Theta\vdash_{t/\{\}}$ P c Q,A
proof −
  from *deriv-strip-triv* **have** *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}$ P c'' *UNIV*,*UNIV*
    by (*auto intro*: *hoare-augment-context*)
  from *deriv-strip* [*simplified* c'']
  have $\Gamma,\Theta\vdash_{/\{\}}$ P (*strip-guards* ($-$ F) c') *UNIV*,*UNIV*
    by (*rule HoarePartialProps*.*MarkGuardsD*)

with *deriv*
have $\Gamma,\Theta\vdash_{t/\{\}}$ *P c′ Q*,*A*
by (*rule CombineStrip*)
hence $\Gamma,\Theta\vdash_{t/\{\}}$ *P mark-guards False c′ Q*,*A*
by (*rule MarkGuardsI*)
hence $\Gamma,\Theta\vdash_{t/\{\}}$ *P merge-guards* (*mark-guards False c′*) *Q*,*A*
by (*rule MergeGuardsI*)
hence $\Gamma,\Theta\vdash_{t/\{\}}$ *P merge-guards c Q*,*A*
by (*simp add*: *c*)
thus *?thesis*
by (*rule MergeGuardsD*)
qed

lemma *CombineStrip′′*:
assumes *deriv*: $\Gamma,\Theta\vdash_{t/\{True\}}$ *P c′ Q*,*A*
assumes *deriv-strip-triv*: $\Gamma,\{\}\vdash_{/\{\}}$ *P c′′ UNIV*,*UNIV*
assumes *c′′*: *c′′*= *mark-guards False* (*strip-guards* ({*False*}) *c′*)
assumes *c*: *merge-guards c* = *merge-guards* (*mark-guards False c′*)
shows $\Gamma,\Theta\vdash_{t/\{\}}$ *P c Q*,*A*
apply (*rule CombineStrip′* [*OF deriv deriv-strip-triv - c*])
apply (*insert c′′*)
apply (*subgoal-tac* − {*True*} = {*False*})
apply *auto*
done

lemma *AsmUN*:
$(\bigcup Z.\ \{(P\ Z,\ p,\ Q\ Z,A\ Z)\}) \subseteq \Theta$
$\Longrightarrow$
$\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ (*P Z*) (*Call p*) (*Q Z*),(*A Z*)
by (*blast intro*: *hoaret.Asm*)

lemma *hoaret-to-hoarep′*:
$\forall Z.\ \Gamma,\{\}\vdash_{t/F}$ (*P Z*) *p* (*Q Z*),(*A Z*) $\Longrightarrow \forall Z.\ \Gamma,\{\}\vdash_{/F}$ (*P Z*) *p* (*Q Z*),(*A Z*)
by (*iprover intro*: *total-to-partial*)

lemma *augment-context′*:
$[\![\Theta \subseteq \Theta'; \forall Z.\ \Gamma,\Theta\vdash_{t/F}$ (*P Z*) *p* (*Q Z*),(*A Z*)$]\!]$
$\Longrightarrow \forall Z.\ \Gamma,\Theta'\vdash_{t/F}$ (*P Z*) *p* (*Q Z*),(*A Z*)
by (*iprover intro*: *hoaret-augment-context*)

lemma *augment-emptyFaults*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{t/\{\}}$ (*P Z*) *p* (*Q Z*),(*A Z*)$]\!] \Longrightarrow$
$\forall Z.\ \Gamma,\{\}\vdash_{t/F}$ (*P Z*) *p* (*Q Z*),(*A Z*)
by (*blast intro*: *augment-Faults*)

lemma *augment-FaultsUNIV*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{t/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)]\!] \Longrightarrow$
$\quad \forall Z.\ \Gamma,\{\}\vdash_{t/UNIV} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
by (*blast intro*: *augment-Faults*)

lemma *PostConjI* [*trans*]:
$[\![\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{t/F} P\ c\ R,B]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ (Q \cap R),(A \cap B)$
by (*rule PostConjI*)

lemma *PostConjI'* :
$[\![\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ R,B]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ (Q \cap R),(A \cap B)$
by (*rule PostConjI*) *iprover*+

lemma *PostConjE* [*consumes 1*]:
assumes *conj*: $\Gamma,\Theta\vdash_{t/F} P\ c\ (Q \cap R),(A \cap B)$
assumes *E*: $[\![\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A;\ \Gamma,\Theta\vdash_{t/F} P\ c\ R,B]\!] \Longrightarrow S$
shows S
proof −
from *conj* **have** $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$ **by** (*rule conseqPost*) *blast*+
moreover
from *conj* **have** $\Gamma,\Theta\vdash_{t/F} P\ c\ R,B$ **by** (*rule conseqPost*) *blast*+
ultimately show S
by (*rule E*)
qed

16.0.1 Rules for Single-Step Proof

We are now ready to introduce a set of Hoare rules to be used in single-step structured proofs in Isabelle/Isar.

Assertions of Hoare Logic may be manipulated in calculational proofs, with the inclusion expressed in terms of sets or predicates. Reversed order is supported as well.

lemma *annotateI* [*trans*]:
$[\![\Gamma,\Theta\vdash_{t/F} P\ anno\ Q,A;\ c = anno]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
by (*simp*)

lemma *annotate-normI*:
assumes *deriv-anno*: $\Gamma,\Theta\vdash_{t/F} P\ anno\ Q,A$
assumes *norm-eq*: *normalize c* = *normalize anno*
shows $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
proof −
from *HoareTotalProps.NormalizeI* [*OF deriv-anno*] *norm-eq*
have $\Gamma,\Theta\vdash_{t/F} P\ normalize\ c\ Q,A$
by *simp*
from *NormalizeD* [*OF this*]

show *?thesis* **.**
qed

lemma *annotateWhile*:
$[\![\Gamma,\Theta\vdash_{t/F} P\ (whileAnnoG\ gs\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (while\ gs\ b\ c)\ Q,A$
by (*simp add*: *whileAnnoG-def*)

lemma *reannotateWhile*:
$[\![\Gamma,\Theta\vdash_{t/F} P\ (whileAnnoG\ gs\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (whileAnnoG\ gs\ b\ J\ V\ c)\ Q,A$
by (*simp add*: *whileAnnoG-def*)

lemma *reannotateWhileNoGuard*:
$[\![\Gamma,\Theta\vdash_{t/F} P\ (whileAnno\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (whileAnno\ b\ J\ V\ c)\ Q,A$
by (*simp add*: *whileAnno-def*)

lemma [*trans*] : $P' \subseteq P \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{t/F} P'\ c\ Q,A$
by (*rule conseqPre*)

lemma [*trans*]: $Q \subseteq Q' \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q',A$
by (*rule conseqPost*) *blast+*

lemma [*trans*]:
$\Gamma,\Theta\vdash_{t/F} \{s.\ P\ s\}\ c\ Q,A \Longrightarrow (\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{t/F} \{s.\ P'\ s\}\ c\ Q,A$
by (*rule conseqPre*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{t/F} \{s.\ P\ s\}\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{t/F} \{s.\ P'\ s\}\ c\ Q,A$
by (*rule conseqPre*) *auto*

lemma [*trans*]:
$\Gamma,\Theta\vdash_{t/F} P\ c\ \{s.\ Q\ s\},A \Longrightarrow (\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ \{s.\ Q\ s\},A \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*intro?*]: $\Gamma,\Theta\vdash_{t/F} P\ Skip\ P,A$
by (*rule Skip*) *auto*

lemma *CondInt* [*trans*,*intro?*]:
$[\![\Gamma,\Theta\vdash_{t/F} (P \cap b)\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F} (P \cap -\ b)\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c1\ c2)\ Q,A$

by (*rule Cond*) *auto*

lemma *CondConj* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{t/F} \{s.\ P\ s \wedge b\ s\}\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F} \{s.\ P\ s \wedge \neg\ b\ s\}\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} \{s.\ P\ s\}\ (Cond\ \{s.\ b\ s\}\ c1\ c2)\ Q,A$
by (*rule Cond*) *auto*
end

17 Auxiliary Definitions/Lemmas to Facilitate Hoare Logic

theory *Hoare* **imports** *HoarePartial HoareTotal* **begin**

syntax

-hoarep-emptyFaults::
[('s,'p,'f) body,('s,'p) quadruple set,
'f set,'s assn,('s,'p,'f) com, 's assn,'s assn] => bool
((3-,-/⊢ (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx::
[('s,'p,'f) body,'f set,'s assn,('s,'p,'f) com, 's assn,'s assn] => bool
((3-/⊢$_{'/-}$ (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx-emptyFaults::
[('s,'p,'f) body,'s assn,('s,'p,'f) com, 's assn,'s assn] => bool
((3-/⊢ (-/ (-)/ -,/-)) [61,1000,20,1000,1000]60)

-hoarep-noAbr::
[('s,'p,'f) body,('s,'p) quadruple set,'f set,
's assn,('s,'p,'f) com, 's assn] => bool
((3-,-/⊢$_{'/-}$ (-/ (-)/ -)) [61,60,60,1000,20,1000]60)

-hoarep-noAbr-emptyFaults::
[('s,'p,'f) body,('s,'p) quadruple set,'s assn,('s,'p,'f) com, 's assn] => bool
((3-,-/⊢ (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoarep-emptyCtx-noAbr::
[('s,'p,'f) body,'f set,'s assn,('s,'p,'f) com, 's assn] => bool
((3-/⊢$_{'/-}$ (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoarep-emptyCtx-noAbr-emptyFaults::
[('s,'p,'f) body,'s assn,('s,'p,'f) com, 's assn] => bool
((3-/⊢ (-/ (-)/ -)) [61,1000,20,1000]60)

-hoaret-emptyFaults::
[('s,'p,'f) *body*,('s,'p) *quadruple set*,
'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] => *bool*
((3-,-/$\vdash_t$ (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoaret-emptyCtx::
[('s,'p,'f) *body*,'*f set*,'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] => *bool*
((3-/$\vdash_{t'/\text{-}}$ (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoaret-emptyCtx-emptyFaults::
[('s,'p,'f) *body*,'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] => *bool*
((3-/$\vdash_t$ (-/ (-)/ -,/-)) [61,1000,20,1000,1000]60)

-hoaret-noAbr::
[('s,'p,'f) *body*,'*f set*, ('s,'p) *quadruple set*,
'*s assn*,('s,'p,'f) *com*, '*s assn*] => *bool*
((3-,-/$\vdash_{t'/\text{-}}$ (-/ (-)/ -)) [61,60,60,1000,20,1000]60)

-hoaret-noAbr-emptyFaults::
[('s,'p,'f) *body*,('s,'p) *quadruple set*,'*s assn*,('s,'p,'f) *com*, '*s assn*] => *bool*
((3-,-/$\vdash_t$ (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoaret-emptyCtx-noAbr::
[('s,'p,'f) *body*,'*f set*,'*s assn*,('s,'p,'f) *com*, '*s assn*] => *bool*
((3-/$\vdash_{t'/\text{-}}$ (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoaret-emptyCtx-noAbr-emptyFaults::
[('s,'p,'f) *body*,'*s assn*,('s,'p,'f) *com*, '*s assn*] => *bool*
((3-/$\vdash_t$ (-/ (-)/ -)) [61,1000,20,1000]60)

syntax (*ASCII*)

-hoarep-emptyFaults::
[('s,'p,'f) *body*,('s,'p) *quadruple set*,
'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] ⇒ *bool*
((3-,-/|− (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx::
[('s,'p,'f) *body*,'*f set*,'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] => *bool*
((3-/|−'/- (-/ (-)/ -,/-)) [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx-emptyFaults::
[('s,'p,'f) *body*,'*s assn*,('s,'p,'f) *com*, '*s assn*,'*s assn*] => *bool*
((3-/|−(-/ (-)/ -,/-)) [61,1000,20,1000,1000]60)

-hoarep-noAbr::
[('s,'p,'f) *body*,('s,'p) *quadruple set*,'*f set*,

$'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-},\text{-}/|-'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,60,1000,20,1000]60)$

-hoarep-noAbr-emptyFaults::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-},\text{-}/|-(\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoarep-emptyCtx-noAbr::
$[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/|-'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoarep-emptyCtx-noAbr-emptyFaults::
$[('s,'p,'f)\ body,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/|-(\text{-}/\ (\text{-})/\ \text{-}))\ [61,1000,20,1000]60)$

-hoaret-emptyFault::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,$
$'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] => bool$
$((3\text{-},\text{-}/|-t\ (\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,60,1000,20,1000,1000]60)$

-hoaret-emptyCtx::
$[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] => bool$
$((3\text{-}/|-t'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,60,1000,20,1000,1000]60)$

-hoaret-emptyCtx-emptyFaults::
$[('s,'p,'f)\ body,'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] => bool$
$((3\text{-}/|-t(\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,1000,20,1000,1000]60)$

-hoaret-noAbr::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-},\text{-}/|-t'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,60,1000,20,1000]60)$

-hoaret-noAbr-emptyFaults::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-},\text{-}/|-t(\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoaret-emptyCtx-noAbr::
$[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/|-t'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoaret-emptyCtx-noAbr-emptyFaults::
$[('s,'p,'f)\ body,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/|-t(\text{-}/\ (\text{-})/\ \text{-}))\ [61,1000,20,1000]60)$

translations

$\Gamma\vdash\ P\ c\ Q,A\ \ == \Gamma\vdash_{/\{\}}\ P\ c\ Q,A$

$\Gamma \vdash_{/F} P\ c\ Q,A\ \ == \Gamma,\{\} \vdash_{/F} P\ c\ Q,A$

$\Gamma,\Theta \vdash\ P\ c\ Q\ \ == \Gamma,\Theta \vdash_{/\{\}}\ P\ c\ Q$
$\Gamma,\Theta \vdash_{/F} P\ c\ Q\ \ == \Gamma,\Theta \vdash_{/F} P\ c\ Q,\{\}$
$\Gamma,\Theta \vdash\ P\ c\ Q,A == \Gamma,\Theta \vdash_{/\{\}}\ P\ c\ Q,A$

$\Gamma \vdash\ P\ c\ Q\quad ==\ \ \Gamma \vdash_{/\{\}}\ P\ c\ Q$
$\Gamma \vdash_{/F} P\ c\ Q\ \ == \Gamma,\{\} \vdash_{/F} P\ c\ Q$
$\Gamma \vdash_{/F} P\ c\ Q\ \ <=\ \ \Gamma \vdash_{/F} P\ c\ Q,\{\}$
$\Gamma \vdash\ P\ c\ Q\quad <=\ \ \Gamma \vdash\ P\ c\ Q,\{\}$

$\Gamma \vdash_t\ P\ c\ Q,A\quad == \Gamma \vdash_{t/\{\}}\ P\ c\ Q,A$
$\Gamma \vdash_{t/F} P\ c\ Q,A\ \ == \Gamma,\{\} \vdash_{t/F} P\ c\ Q,A$

$\Gamma,\Theta \vdash_t\ P\ c\ Q\quad == \Gamma,\Theta \vdash_{t/\{\}}\ P\ c\ Q$
$\Gamma,\Theta \vdash_{t/F} P\ c\ Q == \Gamma,\Theta \vdash_{t/F} P\ c\ Q,\{\}$
$\Gamma,\Theta \vdash_t\ P\ c\ Q,A\quad == \Gamma,\Theta \vdash_{t/\{\}}\ P\ c\ Q,A$

$\Gamma \vdash_t\ P\ c\ Q\quad == \Gamma \vdash_{t/\{\}}\ P\ c\ Q$
$\Gamma \vdash_{t/F} P\ c\ Q\ \ == \Gamma,\{\} \vdash_{t/F} P\ c\ Q$
$\Gamma \vdash_{t/F} P\ c\ Q\ \ <=\ \ \Gamma \vdash_{t/F} P\ c\ Q,\{\}$
$\Gamma \vdash_t\ P\ c\ Q\quad <=\ \ \Gamma \vdash_t\ P\ c\ Q,\{\}$

term $\Gamma \vdash\ P\ c\ Q$
term $\Gamma \vdash\ P\ c\ Q,A$

term $\Gamma \vdash_{/F} P\ c\ Q$
term $\Gamma \vdash_{/F} P\ c\ Q,A$

term $\Gamma,\Theta \vdash\ P\ c\ Q$
term $\Gamma,\Theta \vdash_{/F} P\ c\ Q$

term $\Gamma,\Theta \vdash\ P\ c\ Q,A$
term $\Gamma,\Theta \vdash_{/F} P\ c\ Q,A$

term $\Gamma \vdash_t\ P\ c\ Q$
term $\Gamma \vdash_t\ P\ c\ Q,A$

term $\Gamma \vdash_{t/F} P\ c\ Q$
term $\Gamma \vdash_{t/F} P\ c\ Q,A$

term $\Gamma,\Theta \vdash\ P\ c\ Q$

term $\Gamma,\Theta\vdash_{t/F}$ *P c Q*

term $\Gamma,\Theta\vdash$ *P c Q*,*A*
term $\Gamma,\Theta\vdash_{t/F}$ *P c Q*,*A*

locale *hoare* =
  fixes Γ::('*s*,'*p*,'*f*) *body*

primrec *assoc*:: ('*a* ×'*b*) *list* ⇒ '*a* ⇒ '*b*
where
assoc [] *x* = *undefined* |
assoc (*p*#*ps*) *x* = (*if fst p* = *x then* (*snd p*) *else assoc ps x*)

lemma *conjE-simp*: $(P \wedge Q \Longrightarrow PROP\ R) \equiv (P \Longrightarrow Q \Longrightarrow PROP\ R)$
  by *rule simp-all*

lemma *CollectInt-iff*: $\{s.\ P\ s\} \cap \{s.\ Q\ s\} = \{s.\ P\ s \wedge Q\ s\}$
  by *auto*

lemma *Compl-Collect*:$-(Collect\ b) = \{x.\ \neg(b\ x)\}$
  by *fastforce*

lemma *Collect-False*: $\{s.\ False\} = \{\}$
  by *simp*

lemma *Collect-True*: $\{s.\ True\} = UNIV$
  by *simp*

lemma *triv-All-eq*: $\forall x.\ P \equiv P$
  by *simp*

lemma *triv-Ex-eq*: $\exists x.\ P \equiv P$
  by *simp*

lemma *Ex-True*: $\exists b.\ b$
   by *blast*

lemma *Ex-False*: $\exists b.\ \neg b$
  by *blast*

definition *mex*::('*a* ⇒ *bool*) ⇒ *bool*
  where *mex P* = *Ex P*

definition *meq*::'*a* ⇒ '*a* ⇒ *bool*
  where *meq s Z* = (*s* = *Z*)

lemma *subset-unI1*: $A \subseteq B \Longrightarrow A \subseteq B \cup C$

by *blast*

lemma *subset-unI2*: $A \subseteq C \Longrightarrow A \subseteq B \cup C$
by *blast*

lemma *split-paired-UN*: $(\bigcup p.\ (P\ p)) = (\bigcup a\ b.\ (P\ (a,b)))$
by *auto*

lemma *in-insert-hd*: $f \in insert\ f\ X$
by *simp*

lemma *lookup-Some-in-dom*: $\Gamma\ p = Some\ bdy \Longrightarrow p \in dom\ \Gamma$
by *auto*

lemma *unit-object*: $(\forall u::unit.\ P\ u) = P\ ()$
by *auto*

lemma *unit-ex*: $(\exists u::unit.\ P\ u) = P\ ()$
by *auto*

lemma *unit-meta*: $(\bigwedge(u::unit).\ PROP\ P\ u) \equiv PROP\ P\ ()$
by *auto*

lemma *unit-UN*: $(\bigcup z::unit.\ P\ z) = P\ ()$
by *auto*

lemma *subset-singleton-insert1*: $y = x \Longrightarrow \{y\} \subseteq insert\ x\ A$
by *auto*

lemma *subset-singleton-insert2*: $\{y\} \subseteq A \Longrightarrow \{y\} \subseteq insert\ x\ A$
by *auto*

lemma *in-Specs-simp*: $(\forall x \in \bigcup Z.\ \{(P\ Z,\ p,\ Q\ Z,\ A\ Z)\}.\ Prop\ x) =$
$(\forall Z.\ Prop\ (P\ Z,p,Q\ Z,A\ Z))$
by *auto*

lemma *in-set-Un-simp*: $(\forall x \in A \cup B.\ P\ x) = ((\forall x \in A.\ P\ x) \wedge (\forall x \in B.\ P\ x))$
by *auto*

lemma *split-all-conj*: $(\forall x.\ P\ x \wedge Q\ x) = ((\forall x.\ P\ x) \wedge (\forall x.\ Q\ x))$
by *blast*

lemma *image-Un-single-simp*: $f\ `\ (\bigcup Z.\ \{P\ Z\}) = (\bigcup Z.\ \{f\ (P\ Z)\})$
by *auto*

lemma *measure-lex-prod-def'*:
$f <*mlex*> r \equiv (\{(x,y).\ (x,y) \in measure\ f \vee f\ x=f\ y \wedge (x,y) \in\ r\})$

by (*auto simp add*: *mlex-prod-def inv-image-def*)

lemma *in-measure-iff*: $(x,y) \in measure\ f = (f\ x < f\ y)$
by (*simp add*: *measure-def inv-image-def*)

lemma *in-lex-iff*:
$((a,b),(x,y)) \in r <*lex*> s = ((a,x) \in r \vee (a=x \wedge (b,y)\in s))$
by (*simp add*: *lex-prod-def*)

lemma *in-mlex-iff*:
$(x,y) \in f <*mlex*> r = (f\ x < f\ y \vee (f\ x=f\ y \wedge (x,y) \in r))$
by (*simp add*: *measure-lex-prod-def′ in-measure-iff*)

lemma *in-inv-image-iff*: $(x,y) \in inv\text{-}image\ r\ f = ((f\ x,\ f\ y) \in r)$
by (*simp add*: *inv-image-def*)

This is actually the same as *wf-mlex*. However, this basic proof took me so long that I'm not willing to delete it.

lemma *wf-measure-lex-prod* [*simp*,*intro*]:
assumes *wf-r*: *wf r*
shows $wf\ (f <*mlex*> r)$
proof (*rule ccontr*)
assume $\neg\ wf\ (f <*mlex*> r)$
then
obtain *g* **where** $\forall i.\ (g\ (Suc\ i),\ g\ i) \in f <*mlex*> r$
by (*auto simp add*: *wf-iff-no-infinite-down-chain*)
hence *g*: $\forall i.\ (g\ (Suc\ i),\ g\ i) \in measure\ f \vee$
$f\ (g\ (Suc\ i)) = f\ (g\ i) \wedge (g\ (Suc\ i),\ g\ i) \in r$
by (*simp add*: *measure-lex-prod-def′*)
hence *le-g*: $\forall i.\ f\ (g\ (Suc\ i)) \leq f\ (g\ i)$
by (*auto simp add*: *in-measure-iff order-le-less*)
have *wf* (*measure f*)
by *simp*
hence $\forall Q.\ (\exists x.\ x \in Q) \longrightarrow (\exists z\in Q.\ \forall y.\ (y,\ z) \in measure\ f \longrightarrow y \notin Q)$
by (*simp add*: *wf-eq-minimal*)
from *this* [*rule-format*, *of g ‘ UNIV*]
have $\exists z.\ z \in range\ g \wedge (\forall y.\ (y,\ z) \in measure\ f \longrightarrow y \notin range\ g)$
by *auto*
then obtain *z* **where**
z: $z \in range\ g$ **and**
min-z: $\forall y.\ f\ y < f\ z \longrightarrow y \notin range\ g$
by (*auto simp add*: *in-measure-iff*)
from *z* **obtain** *k* **where**
k: $z = g\ k$
by *auto*
have $\forall i.\ k \leq i \longrightarrow f\ (g\ i) = f\ (g\ k)$
proof (*intro allI impI*)
fix *i*
assume $k \leq i$ **then show** $f\ (g\ i) = f\ (g\ k)$

proof (*induct i*)
case *0*
have $k \leq 0$ **by** *fact* **hence** $k = 0$ **by** *simp*
thus $f\ (g\ 0) = f\ (g\ k)$
by *simp*
next
case (*Suc n*)
have *k-Suc-n*: $k \leq Suc\ n$ **by** *fact*
then show $f\ (g\ (Suc\ n)) = f\ (g\ k)$
proof (*cases* $k = Suc\ n$)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
with *k-Suc-n*
have $k \leq n$
by *simp*
with *Suc.hyps*
have *n-k*: $f\ (g\ n) = f\ (g\ k)$ **by** *simp*
from *le-g* **have** *le*: $f\ (g\ (Suc\ n)) \leq f\ (g\ n)$
by *simp*
show *?thesis*
proof (*cases* $f\ (g\ (Suc\ n)) = f\ (g\ n)$)
case *True* **with** *n-k* **show** *?thesis* **by** *simp*
next
case *False*
with *le* **have** $f\ (g\ (Suc\ n)) < f\ (g\ n)$
by *simp*
with *n-k k* **have** $f\ (g\ (Suc\ n)) < f\ z$
by *simp*
with *min-z* **have** $g\ (Suc\ n) \notin range\ g$
by *blast*
hence *False* **by** *simp*
thus *?thesis*
by *simp*
qed
qed
qed
qed
with *k* [*symmetric*] **have** $\forall i.\ k \leq i \longrightarrow f\ (g\ i) = f\ z$
by *simp*
hence $\forall i.\ k \leq i \longrightarrow f\ (g\ (Suc\ i)) = f\ (g\ i)$
by *simp*
with *g* **have** $\forall i.\ k \leq i \longrightarrow (g\ (Suc\ i),(g\ i)) \in r$
by (*auto simp add*: *in-measure-iff order-less-le*)
hence $\forall i.\ (g\ (Suc\ (i{+}k)),(g\ (i{+}k))) \in r$
by *simp*
then
have $\exists f.\ \forall i.\ (f\ (Suc\ i),\ f\ i) \in r$

by − (*rule exI* [**where** $x=\lambda i.\ g\ (i+k)$],*simp*)
with *wf-r* **show** *False*
by (*simp add*: *wf-iff-no-infinite-down-chain*)
qed

lemmas *all-imp-to-ex* = *all-simps* (*5*)

lemma *all-imp-eq-triv*: $(\forall x.\ x = k \longrightarrow Q) = Q$
$(\forall x.\ k = x \longrightarrow Q) = Q$
by *auto*

end

18 State Space Template

theory *StateSpace* **imports** *Hoare*
begin

record *'g state* = *globals::'g*

definition
upd-globals:: ('g ⇒ 'g) ⇒ ('g,'z) *state-scheme* ⇒ ('g,'z) *state-scheme*
where
upd-globals upd s = *s*(|*globals* := *upd* (*globals s*)|)

record ('g, 'n, 'val) *stateSP* = *'g state* +
locals :: 'n ⇒ 'val

lemma *upd-globals-conv*: *upd-globals f* = (λs. *s*(|*globals* := *f* (*globals s*)|))
by (*rule ext*) (*simp add*: *upd-globals-def*)

end

theory *Generalise* **imports** *HOL−Statespace.DistinctTreeProver*
begin

lemma *protectRefl*: *PROP Pure.prop* (*PROP C*) ⟹ *PROP Pure.prop* (*PROP C*)
by (*simp add*: *prop-def*)

lemma *protectImp*:
assumes *i*: *PROP Pure.prop* (*PROP P* ⟹ *PROP Q*)
shows *PROP Pure.prop* (*PROP Pure.prop P* ⟹ *PROP Pure.prop Q*)
proof −
{

assume P: *PROP Pure.prop P*
from i [*unfolded prop-def*, *OF P* [*unfolded prop-def*]]
have *PROP Pure.prop Q*
by (*simp add*: *prop-def*)
}
note $i' = this$
show *PROP ?thesis*
apply (*rule protectI*)
apply (*rule* i')
apply *assumption*
done
qed

lemma *generaliseConj*:
assumes *i1*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop Q*))
assumes *i2*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* P') $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* Q'))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($P \wedge P'$)) $\Longrightarrow$ (*PROP Pure.prop* (*Trueprop* ($Q \wedge Q'$))))
using *i1 i2*
by (*auto simp add*: *prop-def*)

lemma *generaliseAll*:
assumes i: *PROP Pure.prop* ($\bigwedge s$. *PROP Pure.prop* (*Trueprop* ($P\ s$)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($Q\ s$)))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($\forall s.\ P\ s$)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\forall s.\ Q\ s$)))
using i
by (*auto simp add*: *prop-def*)

lemma *generalise-all*:
assumes i: *PROP Pure.prop* ($\bigwedge s$. *PROP Pure.prop* (*PROP P s*) $\Longrightarrow$ *PROP Pure.prop* (*PROP Q s*))
shows *PROP Pure.prop* ((*PROP Pure.prop* ($\bigwedge s$. *PROP P s*)) $\Longrightarrow$ (*PROP Pure.prop* ($\bigwedge s$. *PROP Q s*)))
using i
proof (*unfold prop-def*)
assume *i1*: $\bigwedge s$. (*PROP P s*) $\Longrightarrow$ (*PROP Q s*)
assume *i2*: $\bigwedge s$. *PROP P s*
show $\bigwedge s$. *PROP Q s*
by (*rule i1*) (*rule i2*)
qed

lemma *generaliseTrans*:
assumes *i1*: *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP Q*)
assumes *i2*: *PROP Pure.prop* (*PROP Q* $\Longrightarrow$ *PROP R*)
shows *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP R*)

using *i1 i2*
proof (*unfold prop-def*)
assume *P-Q*: *PROP P* $\Longrightarrow$ *PROP Q*
assume *Q-R*: *PROP Q* $\Longrightarrow$ *PROP R*
assume *P*: *PROP P*
show *PROP R*
by (*rule Q-R* [*OF P-Q* [*OF P*]])
qed

lemma *meta-spec*:
assumes $\bigwedge x$. *PROP P x*
shows *PROP P x* **by** *fact*

lemma *meta-spec-protect*:
assumes *g*: $\bigwedge x$. *PROP P x*
shows *PROP Pure.prop* (*PROP P x*)
using *g*
by (*auto simp add*: *prop-def*)

lemma *generaliseImp*:
assumes *i*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop Q*))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* (*X* $\longrightarrow$ *P*)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* (*X* $\longrightarrow$ *Q*)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generaliseEx*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s$. *PROP Pure.prop* (*Trueprop* (*P s*)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* (*Q s*)))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($\exists$ *s*. *P s*)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\exists$ *s*. *Q s*)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generaliseRefl*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop P*))
by (*auto simp add*: *prop-def*)

lemma *generaliseRefl'*: *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP P*)
by (*auto simp add*: *prop-def*)

lemma *generaliseAllShift*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s$. *P* $\Longrightarrow$ *Q s*)
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\forall$ *s*. *Q s*)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generalise-allShift*:
assumes *i*: *PROP Pure.prop* ($\bigwedge$*s*. *PROP P* $\Longrightarrow$ *PROP Q s*)
shows *PROP Pure.prop* (*PROP Pure.prop* (*PROP P*) $\Longrightarrow$ *PROP Pure.prop* ($\bigwedge$*s*. *PROP Q s*))
using *i*
proof (*unfold prop-def*)
assume *P-Q*: $\bigwedge$*s*. *PROP P* $\Longrightarrow$ *PROP Q s*
assume *P*: *PROP P*
show $\bigwedge$*s*. *PROP Q s*
by (*rule P-Q* [*OF P*])
qed

lemma *generaliseImpl*:
assumes *i*: *PROP Pure.prop* (*PROP Pure.prop P* $\Longrightarrow$ *PROP Pure.prop Q*)
shows *PROP Pure.prop* ((*PROP Pure.prop* (*PROP X* $\Longrightarrow$ *PROP P*)) $\Longrightarrow$ (*PROP Pure.prop* (*PROP X* $\Longrightarrow$ *PROP Q*)))
using *i*
proof (*unfold prop-def*)
assume *i1*: *PROP P* $\Longrightarrow$ *PROP Q*
assume *i2*: *PROP X* $\Longrightarrow$ *PROP P*
assume *X*: *PROP X*
show *PROP Q*
by (*rule i1* [*OF i2* [*OF X*]])
qed

ML-file *generalise-state.ML*

end

19 Auxiliary Definitions/Lemmas to Facilitate Hoare Logic

theory *HoareCon* **imports** *Main* **begin**

primrec *assoc*:: ('*a* $\times$'*b*) *list* $\Rightarrow$ '*a* $\Rightarrow$ '*b*
where
assoc [] *x* = *undefined* |
assoc (*p*#*ps*) *x* = (*if fst p* = *x then* (*snd p*) *else assoc ps x*)

lemma *conjE-simp*: ($P \wedge Q \Longrightarrow$ *PROP R*) $\equiv$ ($P \Longrightarrow Q \Longrightarrow$ *PROP R*)
by *rule simp-all*

lemma *CollectInt-iff*: $\{s.\ P\ s\} \cap \{s.\ Q\ s\} = \{s.\ P\ s \wedge Q\ s\}$
by *auto*

lemma *Compl-Collect*:$-(Collect\ b) = \{x.\ \neg(b\ x)\}$
by *fastforce*

lemma *Collect-False*: $\{s.\ False\} = \{\}$
by *simp*

lemma *Collect-True*: $\{s.\ True\} = UNIV$
by *simp*

lemma *triv-All-eq*: $\forall x.\ P \equiv P$
by *simp*

lemma *triv-Ex-eq*: $\exists x.\ P \equiv P$
by *simp*

lemma *Ex-True*: $\exists b.\ b$
by *blast*

lemma *Ex-False*: $\exists b.\ \neg b$
by *blast*

definition *mex*::$('a \Rightarrow bool) \Rightarrow bool$
where $mex\ P = Ex\ P$

definition *meq*::$'a \Rightarrow 'a \Rightarrow bool$
where $meq\ s\ Z = (s = Z)$

lemma *subset-unI1*: $A \subseteq B \Longrightarrow A \subseteq B \cup C$
by *blast*

lemma *subset-unI2*: $A \subseteq C \Longrightarrow A \subseteq B \cup C$
by *blast*

lemma *split-paired-UN*: $(\bigcup p.\ (P\ p)) = (\bigcup a\ b.\ (P\ (a,b)))$
by *auto*

lemma *in-insert-hd*: $f \in insert\ f\ X$
by *simp*

lemma *lookup-Some-in-dom*: $\Gamma\ p = Some\ bdy \Longrightarrow p \in dom\ \Gamma$
by *auto*

lemma *unit-object*: $(\forall u{::}unit.\ P\ u) = P\ ()$
by *auto*

lemma *unit-ex*: $(\exists u{::}unit.\ P\ u) = P\ ()$

by *auto*

lemma *unit-meta*: $(\bigwedge(u::unit).\ PROP\ P\ u) \equiv PROP\ P\ ()$
by *auto*

lemma *unit-UN*: $(\bigcup z::unit.\ P\ z) = P\ ()$
by *auto*

lemma *subset-singleton-insert1*: $y = x \Longrightarrow \{y\} \subseteq insert\ x\ A$
by *auto*

lemma *subset-singleton-insert2*: $\{y\} \subseteq A \Longrightarrow \{y\} \subseteq insert\ x\ A$
by *auto*

lemma *in-Specs-simp*: $(\forall x{\in}\bigcup Z.\ \{(P\ Z,\ p,\ Q\ Z,\ A\ Z)\}.\ Prop\ x) =$
$(\forall Z.\ Prop\ (P\ Z,p,Q\ Z,A\ Z))$
by *auto*

lemma *in-set-Un-simp*: $(\forall x{\in}A \cup B.\ P\ x) = ((\forall x \in A.\ P\ x) \wedge (\forall x \in B.\ P\ x))$
by *auto*

lemma *split-all-conj*: $(\forall x.\ P\ x \wedge Q\ x) = ((\forall x.\ P\ x) \wedge (\forall x.\ Q\ x))$
by *blast*

lemma *image-Un-single-simp*: $f\ `\ (\bigcup Z.\ \{P\ Z\}) = (\bigcup Z.\ \{f\ (P\ Z)\})$
by *auto*

lemma *measure-lex-prod-def'*:
$f <*mlex*> r \equiv (\{(x,y).\ (x,y) \in measure\ f \vee f\ x{=}f\ y \wedge (x,y) \in\ r\})$
by (*auto simp add*: *mlex-prod-def inv-image-def*)

lemma *in-measure-iff*: $(x,y) \in measure\ f = (f\ x < f\ y)$
by (*simp add*: *measure-def inv-image-def*)

lemma *in-lex-iff*:
$((a,b),(x,y)) \in r <*lex*> s = ((a,x) \in r \vee (a{=}x \wedge (b,y){\in}s))$
by (*simp add*: *lex-prod-def*)

lemma *in-mlex-iff*:
$(x,y) \in f <*mlex*> r = (f\ x < f\ y \vee (f\ x{=}f\ y \wedge (x,y) \in r))$
by (*simp add*: *measure-lex-prod-def' in-measure-iff*)

lemma *in-inv-image-iff*: $(x,y) \in inv\text{-}image\ r\ f = ((f\ x,\ f\ y) \in r)$
by (*simp add*: *inv-image-def*)

This is actually the same as *wf-mlex*. However, this basic proof took me so long that I'm not willing to delete it.

```
lemma wf-measure-lex-prod [simp,intro]:
  assumes wf-r: wf r
  shows wf (f <*mlex*> r)
proof (rule ccontr)
  assume ¬ wf (f <*mlex*> r)
  then
  obtain g where ∀ i. (g (Suc i), g i) ∈ f <*mlex*> r
    by (auto simp add: wf-iff-no-infinite-down-chain)
  hence g: ∀ i. (g (Suc i), g i) ∈ measure f ∨
    f (g (Suc i)) = f (g i) ∧ (g (Suc i), g i) ∈ r
    by (simp add: measure-lex-prod-def ')
  hence le-g: ∀ i. f (g (Suc i)) ≤ f (g i)
    by (auto simp add: in-measure-iff order-le-less)
  have wf (measure f)
    by simp
  hence  ∀ Q. (∃ x. x ∈ Q) ⟶ (∃ z∈Q. ∀ y. (y, z) ∈ measure f ⟶ y ∉ Q)
    by (simp add: wf-eq-minimal)
  from this [rule-format, of g ` UNIV]
  have ∃ z. z ∈ range g ∧ (∀ y. (y, z) ∈ measure f ⟶ y ∉ range g)
    by auto
  then obtain z where
    z: z ∈ range g and
    min-z: ∀ y. f y < f z ⟶ y ∉ range g
    by (auto simp add: in-measure-iff)
  from z obtain k where
    k: z = g k
    by auto
  have ∀ i. k ≤ i ⟶ f (g i) = f (g k)
  proof (intro allI impI)
    fix i
    assume k ≤ i then show f (g i) = f (g k)
    proof (induct i)
      case 0
      have k ≤ 0 by fact hence k = 0 by simp
      thus f (g 0) = f (g k)
        by simp
    next
      case (Suc n)
      have k-Suc-n: k ≤ Suc n by fact
      then show f (g (Suc n)) = f (g k)
      proof (cases k = Suc n)
        case True
        thus ?thesis by simp
      next
        case False
        with k-Suc-n
        have k ≤ n
          by simp
        with Suc.hyps
```

have *n-k*: $f\ (g\ n) = f\ (g\ k)$ **by** *simp*
from *le-g* **have** *le*: $f\ (g\ (Suc\ n)) \leq f\ (g\ n)$
by *simp*
show *?thesis*
proof (*cases* $f\ (g\ (Suc\ n)) = f\ (g\ n)$)
case *True* **with** *n-k* **show** *?thesis* **by** *simp*
next
case *False*
with *le* **have** $f\ (g\ (Suc\ n)) < f\ (g\ n)$
by *simp*
with *n-k k* **have** $f\ (g\ (Suc\ n)) < f\ z$
by *simp*
with *min-z* **have** $g\ (Suc\ n) \notin range\ g$
by *blast*
hence *False* **by** *simp*
thus *?thesis*
by *simp*
qed
qed
qed
qed
with *k* [*symmetric*] **have** $\forall i.\ k \leq i \longrightarrow f\ (g\ i) = f\ z$
by *simp*
hence $\forall i.\ k \leq i \longrightarrow f\ (g\ (Suc\ i)) = f\ (g\ i)$
by *simp*
with *g* **have** $\forall i.\ k \leq i \longrightarrow (g\ (Suc\ i),(g\ i)) \in r$
by (*auto simp add*: *in-measure-iff order-less-le*)
hence $\forall i.\ (g\ (Suc\ (i{+}k)),(g\ (i{+}k))) \in r$
by *simp*
then
have $\exists f.\ \forall i.\ (f\ (Suc\ i),\ f\ i) \in r$
by − (*rule exI* [**where** $x{=}\lambda i.\ g\ (i{+}k)$],*simp*)
with *wf-r* **show** *False*
by (*simp add*: *wf-iff-no-infinite-down-chain*)
qed

lemmas *all-imp-to-ex* = *all-simps* (*5*)

lemma *all-imp-eq-triv*: $(\forall x.\ x = k \longrightarrow Q) = Q$
$(\forall x.\ k = x \longrightarrow Q) = Q$
by *auto*

end
theory *VcgCommon*
imports *../EmbSimpl/StateSpace HOL−Statespace.StateSpaceLocale ../EmbSimpl/Generalise ../EmbSimpl/HoareCon*

begin

definition *list-multsel*:: *'a list* ⇒ *nat list* ⇒ *'a list* (**infixl** !! *100*)
  where *xs* !! *ns* = *map* (*nth xs*) *ns*

definition *list-multupd*:: *'a list* ⇒ *nat list* ⇒ *'a list* ⇒ *'a list*
  where *list-multupd xs ns ys* = *foldl* (λ*xs* (*n*,*v*). *xs*[*n*:=*v*]) *xs* (*zip ns ys*)

nonterminal *lmupdbinds* **and** *lmupdbind*

syntax
  — @ multiple list update
  -lmupdbind:: [*'a*, *'a*] => *lmupdbind*   ((*2*- [:=]/ -))
  :: *lmupdbind* => *lmupdbinds*   (-)
  -lmupdbinds :: [*lmupdbind*, *lmupdbinds*] => *lmupdbinds*   (-,/ -)
  -LMUpdate :: [*'a*, *lmupdbinds*] => *'a*   (-/[(-)] [*900*,*0*] *900*)

translations
  -LMUpdate xs (*-lmupdbinds b bs*) == *-LMUpdate* (*-LMUpdate xs b*) *bs*
  xs[*is*[:=]*ys*] == *CONST list-multupd xs is ys*

reverse application

definition *rapp*:: *'a* ⇒ (*'a* ⇒ *'b*) ⇒ *'b* (**infixr** |> *60*)
  where *rapp x f* = *f x*

nonterminal
  bdy **and**
  newinit **and**
  newinits **and**
  grds **and**
  grd **and**
  locinit **and**
  locinits **and**
  basics **and**
  basic **and**
  basicblock **and**
  switchcase **and**
  switchcases

syntax
  -quote   :: *'b* => (*'a* => *'b*)
  -antiquoteCur0 :: (*'a* => *'b*) => *'b*   (´- [*1000*] *1000*)
  -antiquoteCur :: (*'a* => *'b*) => *'b*
  -antiquoteOld0 :: (*'a* => *'b*) => *'a* => *'b*   (ˉ- [*1000*,*1000*] *1000*)
  -antiquoteOld :: (*'a* => *'b*) => *'a* => *'b*
  -Assert   :: *'a* => *'a set*   (({|-|}) [*0*] *1000*)
  -AssertState :: *idt* ⇒ *'a* => *'a set*   (({|-. -|}) [*1000*,*0*] *1000*)
  -guarantee   :: *'s set* ⇒ *grd*   (-√ [*1000*] *1000*)
  -guaranteeStrip:: *'s set* ⇒ *grd*   (-# [*1000*] *1000*)
  -grd   :: *'s set* ⇒ *grd*   (- [*1000*] *1000*)

```
  -last-grd      :: grd ⇒ grds          (- 1000)
  -grds          :: [grd, grds] ⇒ grds (-,/ - [999,1000] 1000)
  -newinit       :: [ident,'a] ⇒ newinit ((2´- :==/ -))
               :: newinit ⇒ newinits    (-)
  -newinits     :: [newinit, newinits] ⇒ newinits (-,/ -)
  -locnoinit    :: ident ⇒ locinit               (´-)
  -locinit      :: [ident,'a] ⇒ locinit          ((2´- :==/ -))
               :: locinit ⇒ locinits             (-)
  -locinits     :: [locinit, locinits] ⇒ locinits (-,/ -)
  -BasicBlock:: basics ⇒ basicblock (-)
  -BAssign   :: 'b => 'b => basic    ((- :==/ -) [30, 30] 23)
             :: basic ⇒ basics              (-)
  -basics    :: [basic, basics] ⇒ basics (-,/ -)
  -switchcasesSingle  :: switchcase ⇒ switchcases (-)
  -switchcasesCons:: switchcase ⇒ switchcases ⇒ switchcases
                    (-/ | -)
syntax (ASCII)
  -Assert      :: 'a => 'a set           (({|-|}) [0] 1000)
  -AssertState :: idt ⇒ 'a ⇒ 'a set    (({|-. -|}) [1000,0] 1000)

syntax (xsymbols)
  -Assert      :: 'a => 'a set            ((⦃-⦄) [0] 1000)
  -AssertState :: idt ⇒ 'a => 'a set     ((⦃-. -⦄) [1000,0] 1000)
  -AssertR      :: 'a => 'a set            ((⦃-⦄r) [0] 1000)

translations
 (-switchcasesSingle b) => [b]
 (-switchcasesCons b bs) => CONST Cons b bs

parse-ast-translation ‹
  let
    fun tr c asts = Ast.mk-appl (Ast.Constant c) (map Ast.strip-positions asts)
  in
   [(@{syntax-const -antiquoteCur0}, K (tr @{syntax-const -antiquoteCur})),
    (@{syntax-const -antiquoteOld0}, K (tr @{syntax-const -antiquoteOld}))]
  end
›

print-ast-translation ‹
  let
    fun tr c asts = Ast.mk-appl (Ast.Constant c) asts
  in
   [(@{syntax-const -antiquoteCur}, K (tr @{syntax-const -antiquoteCur0})),
    (@{syntax-const -antiquoteOld}, K (tr @{syntax-const -antiquoteOld0}))]
  end
›

nonterminal par and pars and actuals
```

syntax
-*par* :: $'a \Rightarrow$ *par* (-)
:: *par* $\Rightarrow$ *pars* (-)
-*pars* :: [*par*,*pars*] $\Rightarrow$ *pars* (-,/-)
-*actuals* :: *pars* $\Rightarrow$ *actuals* ('(-'))
-*actuals-empty* :: *actuals* ('('))

syntax
-*faccess* :: $'ref \Rightarrow ('ref \Rightarrow 'v) \Rightarrow 'v$
(-→- [65,1000] 100)

syntax (*ASCII*)
-*faccess* :: $'ref \Rightarrow ('ref \Rightarrow 'v) \Rightarrow 'v$
(-−>- [65,1000] 100)

translations

p→*f* => *f p*
g→(-*antiquoteCur f*) <= -*antiquoteCur f g*
{|*s*. *P*|} == {|-*antiquoteCur*((=) *s*) ∧ *P* |}
{|*b*|} => *CONST Collect* (-*quote b*)

nonterminal *modifyargs*

syntax
-*may-modify* :: ['*a*,'*a*,*modifyargs*] $\Rightarrow$ *bool*
(- *may'-only'-modify'-globals* - *in* [-] [100,100,0] 100)
-*may-not-modify* :: ['*a*,'*a*] $\Rightarrow$ *bool*
(- *may'-not'-modify'-globals* - [100,100] 100)
-*may-modify-empty* :: ['*a*,'*a*] $\Rightarrow$ *bool*
(- *may'-only'-modify'-globals* - *in* [] [100,100] 100)
-*modifyargs* :: [*id*,*modifyargs*] $\Rightarrow$ *modifyargs* (-,/ -)
:: *id* => *modifyargs* (-)

translations
s may-only-modify-globals Z in [] => *s may-not-modify-globals Z*
axiomatization *NoBody*::('*s*,'*p*,'*f*) *com*

ML-file *hoare.ML*
ML-file *hoare-syntax.ML*

parse-translation ‹
let
val argsC = @{*syntax-const -modifyargs*};
val globalsN = *globals*;
val ex = @{*const-syntax mex*};
val eq = @{*const-syntax meq*};
val varn = *Hoare-Con.varname*;

```
fun extract-args (Const (argsC,-)$Free (n,-)$t) = varn n::extract-args t
  | extract-args (Free (n,-)) = [varn n]
  | extract-args t             = raise TERM (extract-args, [t])

fun idx [] y = error idx: element not in list
 |  idx (x::xs) y = if x=y then 0 else (idx xs y)+1

fun gen-update ctxt names (name,t) =
      Hoare-Syntax-Common.update-comp ctxt [] false true name (Bound (idx
names name)) t

fun gen-updates ctxt names t = Library.foldr (gen-update ctxt names) (names,t)


fun gen-ex (name,t) = Syntax.const ex $ Abs (name,dummyT,t)

fun gen-exs names t = Library.foldr gen-ex (names,t)


fun tr ctxt s Z names =
  let val upds = gen-updates ctxt (rev names) (Syntax.free globalsN$Z);
      val eq   = Syntax.const eq $ (Syntax.free globalsN$s) $ upds;
  in gen-exs names eq end;

fun may-modify-tr ctxt [s,Z,names] = tr ctxt s Z
                                   (sort-strings (extract-args names))
fun may-not-modify-tr ctxt [s,Z] = tr ctxt s Z []
in
 [(@{syntax-const -may-modify}, may-modify-tr),
  (@{syntax-const -may-not-modify}, may-not-modify-tr)]
end
›
```

print-translation ‹

```
let
  val argsC = @{syntax-const -modifyargs};
  val chop = Hoare-Con.chopsfx Hoare-Con.deco;

  fun get-state ( - $ - $ t) = get-state t  (* for record−updates*)
    | get-state ( - $ - $ - $ - $ - $ t) = get-state t (* for statespace−updates *)
    | get-state (globals$(s as Const (@{syntax-const -free},-) $ Free -)) = s
    | get-state (globals$(s as Const (@{syntax-const -bound},-) $ Free -)) = s
    | get-state (globals$(s as Const (@{syntax-const -var},-) $ Var -)) = s
    | get-state (globals$(s as Const -)) = s
    | get-state (globals$(s as Free -)) = s
    | get-state (globals$(s as Bound -)) = s
    | get-state t              = raise Match;
```

```
  fun mk-args [n] = Syntax.free (chop n)
    | mk-args (n::ns) = Syntax.const argsC $ Syntax.free (chop n) $ mk-args ns
    | mk-args -       = raise Match;

  fun tr' names (Abs (n,-,t)) = tr' (n::names) t
    | tr' names (Const (@{const-syntax mex},-) $ t) = tr' names t
    | tr' names (Const (@{const-syntax meq},-) $ (globals$s) $ upd) =
        let val Z = get-state upd;

        in (case names of
              [] => Syntax.const @{syntax-const -may-not-modify} $ s $ Z
            | xs => Syntax.const @{syntax-const -may-modify} $ s $ Z $ mk-args
(rev names))
        end;

  fun may-modify-tr' [t] = tr' [] t
  fun may-not-modify-tr' [-$s,-$Z] = Syntax.const @{syntax-const -may-not-modify}
$ s $ Z
 in
  [(@{const-syntax mex}, K may-modify-tr'),
   (@{const-syntax meq}, K may-not-modify-tr')]
 end
›
```

syntax
```
-Measure:: ('a ⇒ nat) ⇒ ('a × 'a) set
      (MEASURE - [22] 1)
-Mlex:: ('a ⇒ nat) ⇒ ('a × 'a) set ⇒ ('a × 'a) set
      (infixr <*MLEX*> 30)
-to-quote:: 'b ⇒ ('a ⇒ 'b)
      (quot - [22] 1)

-to-anti-quote:: ('a ⇒ 'b) ⇒ 'b
      (antiquot - [22] 1)
```

translations
```
 MEASURE f        => (CONST measure) (-quote f)
 f <*MLEX*> r       => (-quote f) <*mlex*> r
 quot P        => (-quote P)
 antiquot P =>  (-antiquoteCur P)
```

print-translation ‹
```
 let
  fun selector (Const (c,T)) = Hoare-Con.is-state-var c
    | selector - = false;

  fun measure-tr' ctxt ((t as (Abs (-,-,p)))::ts) =
       if Hoare-Syntax-Common.antiquote-applied-only-to selector p
     then Hoare-Syntax-Common.app-quote-tr' ctxt (Syntax.const @{syntax-const
```

-Measure}) (*t*::*ts*)
          else raise Match
      | *measure-tr′ - - = raise Match*

    fun mlex-tr′ ctxt ((*t as* (*Abs* (-,-,*p*)))::*r*::*ts*) =
          if Hoare-Syntax-Common.antiquote-applied-only-to selector p
       then Hoare-Syntax-Common.app-quote-tr′ ctxt (*Syntax.const* @{*syntax-const*
-Mlex}) (*t*::*r*::*ts*)
          else raise Match
      | *mlex-tr′ - - = raise Match*

  in
   [(@{*const-syntax measure*}, *measure-tr′*),
    (@{*const-syntax mlex-prod*}, *mlex-tr′*)]
  end
›

parse-translation ‹
  let
     fun quote-tr1 ctxt [*t*] = *Hoare-Syntax-Common.quote-tr ctxt* @{*syntax-const*
-antiquoteCur} *t*
      | *quote-tr1 ctxt ts = raise TERM* (*quote-tr1*, *ts*);
  in [(@{*syntax-const -quote*}, *quote-tr1*)] *end*
›

parse-translation ‹
 [(@{*syntax-const -antiquoteCur*},
   K (*Hoare-Syntax-Common.antiquote-varname-tr* @{*syntax-const -antiquoteCur*}))]
›

parse-translation ‹
 [(@{*syntax-const -antiquoteOld*}, *Hoare-Syntax-Common.antiquoteOld-tr*),
  (@{*syntax-const -BasicBlock*}, *Hoare-Syntax-Common.basic-assigns-tr*)]
›

end

20 Facilitating the Hoare Logic

theory *VcgCon*
imports *common*/*VcgCommon LocalRG-HoareDef*
keywords *procedures hoarestate* :: *thy-decl*
begin

locale *hoare* =
  fixes Γ::(′*s*,′*p*,′*f*,′*e*) *body*

axiomatization *NoBody*::(′*s*,′*p*,′*f*,′*e*) *com*

ML-file *hoare.ML*

Variables of the programming language are represented as components of a record. To avoid cluttering up the namespace of Isabelle with lots of typical variable names, we append a unusual suffix at the end of each name by parsing

definition *to-normal*::'*a* ⇒ '*a* ⇒ ('*a*, '*b*) *xstate* × ('*a*, '*b*) *xstate*
where
to-normal a b ≡ (*Normal a*,*Normal b*)

20.1 Some Fancy Syntax

reverse application

definition *rapp*:: '*a* ⇒ ('*a* ⇒ '*b*) ⇒ '*b* (**infixr** |> *60*)
where *rapp x f* = *f x*

notation
Skip (*SKIP*) **and**
Throw (*THROW*)

syntax
-*raise*:: '*c* ⇒ '*c* ⇒ ('*a*,'*b*,'*f*,'*e*) *com* ((*RAISE* - :==/ -) [*30*, *30*] *23*)
-*raise-ev*:: '*c* ⇒ '*e* ⇒ '*c* ⇒ ('*a*,'*b*,'*f*,'*e*) *com* ((*RAISE* - :==(_)/ -) [*30*, *30*, *30*] *23*)
-*seq*::('*s*,'*p*,'*f*,'*e*) *com* ⇒ ('*s*,'*p*,'*f*,'*e*) *com* ⇒ ('*s*,'*p*,'*f*,'*e*) *com* (-;;/ - [*20*, *21*] *20*)
-*guarantee* :: '*s set* ⇒ *grd* (-√ [*1000*] *1000*)
-*guaranteeStrip*:: '*s set* ⇒ *grd* (-# [*1000*] *1000*)
-*grd* :: '*s set* ⇒ *grd* (- [*1000*] *1000*)
-*last-grd* :: *grd* ⇒ *grds* (- *1000*)
-*grds* :: [*grd*, *grds*] ⇒ *grds* (-,/ - [*999*,*1000*] *1000*)
-*guards* :: *grds* ⇒ ('*s*,'*p*,'*f*,'*e*) *com* ⇒ ('*s*,'*p*,'*f*,'*e*) *com*
((-/⟼ -) [*60*, *21*] *23*)

-*Normal* :: '*a* => '*b*

-*Assign* :: '*b* => '*b* => ('*s*,'*p*,'*f*,'*e*) *com* ((- :==/ -) [*30*, *30*] *23*)
-*Assign-ev* :: '*b* => '*e* ⇒'*b* ⇒ ('*s*,'*p*,'*f*,'*e*) *com* ((- :==(_)/ -) [*30*,*1000*, *30*] *23*)
-*Init* :: *ident* ⇒ '*c* ⇒ '*b* ⇒ ('*s*,'*p*,'*f*,'*e*) *com*
((´- :==_/ -) [*30*,*1000*, *30*] *23*)
-*Init-ev* :: *ident* ⇒ '*c* ⇒ '*e* ⇒ '*b* ⇒ ('*s*,'*p*,'*f*,'*e*) *com*
((´- :==(_)/ _/ -) [*30*,*1000*, *1000*,*30*] *23*)
-*GuardedAssign*:: '*b* => '*b* => ('*s*,'*p*,'*f*,'*e*) *com* ((- :==$_g$/ -) [*30*, *30*] *23*)
-*GuardedAssign-ev*:: '*b* => '*e* ⇒ '*b* => ('*s*,'*p*,'*f*,'*e*) *com* ((- :==$_g$-/ -) [*30*, *30*, *30*] *23*)

-New :: ['a, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==/(2 NEW -/ [-])) [30, 65, 0] 23)
-New-ev :: ['a, 'e, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==(_)/(2 NEW -/ [-])) [30, 30, 65, 0] 23)
-GuardedNew :: ['a, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==$_g$/(2 NEW -/ [-])) [30, 65, 0] 23)
-GuardedNew-ev :: ['a,'e, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==$_{g\text{-}}$/(2 NEW -/ [-])) [30, 30, 65, 0] 23)
-NNew :: ['a, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==/(2 NNEW -/ [-])) [30, 65, 0] 23)
-NNew-ev :: ['a, 'e, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==(_)/(2 NNEW -/ [-])) [30, 30, 65, 0] 23)
-GuardedNNew :: ['a, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==$_g$/(2 NNEW -/ [-])) [30, 65, 0] 23)
-GuardedNNew-ev :: ['a, 'e, 'b, newinits] ⇒ ('a,'b,'f,'e) com
((- :==$_{g\text{-}}$/(2 NNEW -/ [-])) [30, 30, 65, 0] 23)

-Cond :: 'a bexp => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com
((0IF (-)/ (2THEN/ -)/ (2ELSE -)/ FI) [0, 0, 0] 71)
-Cond-no-else:: 'a bexp => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com
((0IF (-)/ (2THEN/ -)/ FI) [0, 0] 71)
-GuardedCond :: 'a bexp => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com
((0IF$_g$ (-)/ (2THEN -)/ (2ELSE -)/ FI) [0, 0, 0] 71)
-GuardedCond-no-else:: 'a bexp => ('s,'p,'f,'e) com => ('s,'p,'f,'e) com
((0IF$_g$ (-)/ (2THEN -)/ FI) [0, 0] 71)
-Await :: 'a bexp ⇒ ('s,'p,'f,'e) com ⇒('s,'p,'f,'e) com
((0AWAIT (-)/ -) [0, 0] 71)
-Await-ev :: 'e ⇒ 'a bexp ⇒ ('s,'p,'f,'e) com ⇒ ('s,'p,'f,'e) com
((0AWAIT$_{\downarrow\text{-}}$ (-)/ -) [0,0, 0] 71)
-GuardedAwait :: 'a bexp ⇒ ('s,'p,'f,'e) com ⇒('s,'p,'f,'e) com
((0AWAIT$_g$ (-)/ -) [0, 0] 71)
-GuardedAwait-ev :: 'e ⇒ 'a bexp ⇒ ('s,'p,'f,'e) com ⇒('s,'p,'f,'e) com
((0AWAIT$_{g\downarrow\text{-}}$ (-)/ -) [0,0, 0] 71)
-While-inv-var :: 'a bexp => 'a assn ⇒ ('a × 'a) set ⇒ bdy
⇒ ('s,'p,'f,'e) com
((0WHILE (-)/ INV (-)/ VAR (-) /-) [25, 0, 0, 81] 71)
-WhileFix-inv-var :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒
('z ⇒ ('a × 'a) set) ⇒ bdy
⇒ ('s,'p,'f,'e) com
((0WHILE (-)/ FIX -./ INV (-)/ VAR (-) /-) [25, 0, 0, 0, 81] 71)
-WhileFix-inv :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒ bdy
⇒ ('s,'p,'f,'e) com
((0WHILE (-)/ FIX -./ INV (-) /-) [25, 0, 0, 81] 71)
-GuardedWhileFix-inv-var :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒
('z ⇒ ('a × 'a) set) ⇒ bdy
⇒ ('s,'p,'f,'e) com
((0WHILE$_g$ (-)/ FIX -./ INV (-)/ VAR (-) /-) [25, 0, 0, 0, 81] 71)

-*GuardedWhileFix-inv-var-hook* :: $'a$ *bexp* $\Rightarrow$ ($'z \Rightarrow$ $'a$ *assn*) $\Rightarrow$
($'z \Rightarrow$ ($'a \times$ $'a$) *set*) $\Rightarrow$ *bdy*
$\Rightarrow$ ($'s,'p,'f,'e$) *com*
-*GuardedWhileFix-inv* :: $'a$ *bexp* => *pttrn* $\Rightarrow$ ($'z \Rightarrow$ $'a$ *assn*) $\Rightarrow$ *bdy*
$\Rightarrow$ ($'s,'p,'f,'e$) *com*
(($0WHILE_g$ (-)/ *FIX* -./ *INV* (-)/-) [*25*, *0*, *0*, *81*] *71*)

-*GuardedWhile-inv-var*::
$'a$ *bexp* => $'a$ *assn* $\Rightarrow$ ($'a \times$ $'a$) *set* $\Rightarrow$ *bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
(($0WHILE_g$ (-)/ *INV* (-)/ *VAR* (-) /-) [*25*, *0*, *0*, *81*] *71*)
-*While-inv* :: $'a$ *bexp* => $'a$ *assn* => *bdy* => ($'s,'p,'f,'e$) *com*
((*0WHILE* (-)/ *INV* (-) /-) [*25*, *0*, *81*] *71*)
-*GuardedWhile-inv* :: $'a$ *bexp* => $'a$ *assn* => ($'s,'p,'f,'e$) *com* => ($'s,'p,'f,'e$) *com*
(($0WHILE_g$ (-)/ *INV* (-) /-) [*25*, *0*, *81*] *71*)
-*While* :: $'a$ *bexp* => *bdy* => ($'s,'p,'f,'e$) *com*
((*0WHILE* (-) /-) [*25*, *81*] *71*)
-*GuardedWhile* :: $'a$ *bexp* => *bdy* => ($'s,'p,'f,'e$) *com*
(($0WHILE_g$ (-) /-) [*25*, *81*] *71*)
-*While-guard* :: *grds* => $'a$ *bexp* => *bdy* => ($'s,'p,'f,'e$) *com*
((*0WHILE* (-/$\longmapsto$ (*1*-)) /-) [*1000*,*25*,*81*] *71*)
-*While-guard-inv*:: *grds* $\Rightarrow$$'a$ *bexp*$\Rightarrow$$'a$ *assn*$\Rightarrow$*bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0WHILE* (-/$\longmapsto$ (*1*-)) *INV* (-) /-) [*1000*,*25*,*0*,*81*] *71*)
-*While-guard-inv-var*:: *grds* $\Rightarrow$$'a$ *bexp*$\Rightarrow$$'a$ *assn*$\Rightarrow$($'a$$\times$$'a$) *set*
$\Rightarrow$*bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0WHILE* (-/$\longmapsto$ (*1*-)) *INV* (-)/ *VAR* (-) /-) [*1000*,*25*,*0*,*0*,*81*] *71*)
-*WhileFix-guard-inv-var*:: *grds* $\Rightarrow$$'a$ *bexp*$\Rightarrow$*pttrn*$\Rightarrow$($'z$$\Rightarrow$$'a$ *assn*)$\Rightarrow$($'z$$\Rightarrow$($'a$$\times$$'a$) *set*)
$\Rightarrow$*bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0WHILE* (-/$\longmapsto$ (*1*-)) *FIX* -./ *INV* (-)/ *VAR* (-) /-) [*1000*,*25*,*0*,*0*,*0*,*81*] *71*)
-*WhileFix-guard-inv*:: *grds* $\Rightarrow$$'a$ *bexp*$\Rightarrow$*pttrn*$\Rightarrow$($'z$$\Rightarrow$$'a$ *assn*)
$\Rightarrow$*bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0WHILE* (-/$\longmapsto$ (*1*-)) *FIX* -./ *INV* (-)/-) [*1000*,*25*,*0*,*0*,*81*] *71*)

-*Try-Catch*:: ($'s,'p,'f,'e$) *com* $\Rightarrow$($'s,'p,'f,'e$) *com* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0TRY* (-)/ (*2CATCH* -)/ *END*) [*0*,*0*] *71*)

-*DoPre* :: ($'s,'p,'f,'e$) *com* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
-*Do* :: ($'s,'p,'f,'e$) *com* $\Rightarrow$ *bdy* ((*2DO*/ (-)) /*OD* [*0*] *1000*)
-*Lab*:: $'a$ *bexp* $\Rightarrow$ ($'s,'p,'f,'e$) *com* $\Rightarrow$ *bdy*
(-•/- [*1000*,*71*] *81*)
:: *bdy* $\Rightarrow$ ($'s,'p,'f,'e$) *com* (-)
-*Spec*:: *pttrn* $\Rightarrow$ $'s$ *set* $\Rightarrow$ ($'s,'p,'f,'e$) *com* $\Rightarrow$ $'s$ *set* $\Rightarrow$ $'s$ *set* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*ANNO* -. -/ (-)/ -,/-) [*0*,*1000*,*20*,*1000*,*1000*] *60*)
-*SpecNoAbrupt*:: *pttrn* $\Rightarrow$ $'s$ *set* $\Rightarrow$ ($'s,'p,'f,'e$) *com* $\Rightarrow$ $'s$ *set* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*ANNO* -. -/ (-)/ -) [*0*,*1000*,*20*,*1000*] *60*)
-*LemAnno*:: $'n$ $\Rightarrow$ ($'s,'p,'f,'e$) *com* $\Rightarrow$ ($'s,'p,'f,'e$) *com*
((*0 LEMMA* (-)/ - *END*) [*1000*,*0*] *71*)

-Loc:: [*locinits*,(*'s*,*'p*,*'f*,*'e*) *com*] ⇒ (*'s*,*'p*,*'f*,*'e*) *com*
((*2 LOC -;;/ (-) COL*) [*0*,*0*] *71*)
-Switch:: (*'s* ⇒ *'v*) ⇒ *switchcases* ⇒ (*'s*,*'p*,*'f*,*'e*) *com*
((*0 SWITCH* (-)/ - *END*) [*22*,*0*] *71*)
-switchcase:: *'v set* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ⇒ *switchcase* (-⇒/ -)

-Basic:: *basicblock* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ((*0BASIC*/ (-)/ *END*) [*22*] *71*)
-Basic-ev:: *'e* ⇒ *basicblock* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ((*0BASIC*(_)/ (-)/ *END*) [*22*, *22*] *71*)

syntax (*ascii*)
-While-guard :: *grds* => *'a bexp* => *bdy* ⇒ (*'s*,*'p*,*'f*,*'e*) *com*
((*0WHILE* (-|−> /-) /-) [*0*,*0*,*1000*] *71*)
-While-guard-inv:: *grds*⇒*'a bexp*⇒*'a assn*⇒*bdy* ⇒ (*'s*,*'p*,*'f*,*'e*) *com*
((*0WHILE* (-|−> /-) *INV* (-) /-) [*0*,*0*,*0*,*1000*] *71*)
-guards :: *grds* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ⇒ (*'s*,*'p*,*'f*,*'e*) *com* ((-|−>-) [*60*, *21*] *23*)

syntax (**output**)
-hidden-grds :: *grds* (. . .)

translations
-Do c => *c*
b· *c* => *CONST condCatch c b SKIP*
b· (*-DoPre c*) <= *CONST condCatch c b SKIP*
l· (*CONST whileAnnoG gs b I V c*) <= *l*· (*-DoPre* (*CONST whileAnnoG gs b I V c*))
l· (*CONST whileAnno b I V c*) <= *l*· (*-DoPre* (*CONST whileAnno b I V c*))
CONST condCatch c b SKIP <= (*-DoPre* (*CONST condCatch c b SKIP*))
-Do c <= *-DoPre c*
c;; *d* == *CONST Seq c d*
-guarantee g => (*CONST True*, *g*)
-guaranteeStrip g == *CONST guaranteeStripPair* (*CONST True*) *g*
-grd g => (*CONST False*, *g*)
-grds g gs => *g*#*gs*
-last-grd g => [*g*]
-guards gs c == *CONST guards gs c*

IF b THEN c1 ELSE c2 FI => *CONST Cond* {|*b*|} *c1 c2*
IF b THEN c1 FI == *IF b THEN c1 ELSE SKIP FI*
IF_g *b THEN c1 FI* == IF_g *b THEN c1 ELSE SKIP FI*

AWAIT b c == *CONST Await* {|*b*|} *c* (*CONST None*)
$AWAIT_{\downarrow e}$ *b c* == *CONST Await* {|*b*|} *c* (*CONST Some e*)

-*While-inv-var b I V c* => *CONST whileAnno* {|*b*|} *I V c*
-*While-inv-var b I V* (-*DoPre c*) <= *CONST whileAnno* {|*b*|} *I V c*
-*While-inv b I c* == -*While-inv-var b I* (*CONST undefined*) *c*
-*While b c* == -*While-inv b* {|*CONST undefined*|} *c*

-*While-guard-inv-var gs b I V c* => *CONST whileAnnoG gs* {|*b*|} *I V c*

-*While-guard-inv gs b I c* == -*While-guard-inv-var gs b I* (*CONST undefined*) *c*
-*While-guard gs b c* == -*While-guard-inv gs b* {|*CONST undefined*|} *c*

-*GuardedWhile-inv b I c* == -*GuardedWhile-inv-var b I* (*CONST undefined*) *c*
-*GuardedWhile b c* == -*GuardedWhile-inv b* {|*CONST undefined*|} *c*

TRY c1 CATCH c2 END == *CONST Catch c1 c2*
ANNO s. P c Q,*A* => *CONST specAnno* (λ*s. P*) (λ*s. c*) (λ*s. Q*) (λ*s. A*)
ANNO s. P c Q == *ANNO s. P c Q*,{}

-*WhileFix-inv-var b z I V c* => *CONST whileAnnoFix* {|*b*|} (λ*z. I*) (λ*z. V*) (λ*z. c*)
-*WhileFix-inv-var b z I V* (-*DoPre c*) <= -*WhileFix-inv-var* {|*b*|} *z I V c*
-*WhileFix-inv b z I c* == -*WhileFix-inv-var b z I* (*CONST undefined*) *c*

-*GuardedWhileFix-inv b z I c* == -*GuardedWhileFix-inv-var b z I* (*CONST undefined*) *c*

-*GuardedWhileFix-inv-var b z I V c* =>
-*GuardedWhileFix-inv-var-hook* {|*b*|} (λ*z. I*) (λ*z. V*) (λ*z. c*)

-*WhileFix-guard-inv-var gs b z I V c* =>
CONST whileAnnoGFix gs {|*b*|} (λ*z. I*) (λ*z. V*) (λ*z. c*)
-*WhileFix-guard-inv-var gs b z I V* (-*DoPre c*) <=
-*WhileFix-guard-inv-var gs* {|*b*|} *z I V c*
-*WhileFix-guard-inv gs b z I c* == -*WhileFix-guard-inv-var gs b z I* (*CONST undefined*) *c*
LEMMA x c END == *CONST lem x c*

translations
(-*switchcase V c*) => (*V*,*c*)

(-*Switch v vs*) => *CONST switch* (-*quote v*) *vs*

print-ast-translation ⟪
let
fun dest-abs (*Ast.Appl* [*Ast.Constant* @{*syntax-const -abs*}, *x*, *t*]) = (*x*, *t*)
| *dest-abs* - = *raise Match*;

fun spec-tr′ [P, c, Q, A] =
let
val (x',P') = *dest-abs* P;
val (- ,c') = *dest-abs* c;
val (- ,Q') = *dest-abs* Q;
val (- ,A') = *dest-abs* A;
in
if (A' = *Ast.Constant* @{*const-syntax bot*})
then Ast.mk-appl (*Ast.Constant* @{*syntax-const -SpecNoAbrupt*}) [x', P', c', Q']
else Ast.mk-appl (*Ast.Constant* @{*syntax-const -Spec*}) [x', P', c', Q', A']
end;
fun whileAnnoFix-tr′ [b, I, V, c] =
let
val (x',I') = *dest-abs* I;
val (- ,V') = *dest-abs* V;
val (- ,c') = *dest-abs* c;
in
Ast.mk-appl (*Ast.Constant* @{*syntax-const -WhileFix-inv-var*}) [b, x', I', V', c']
end;
in
[(@{*const-syntax specAnno*}, *K spec-tr′*),
(@{*const-syntax whileAnnoFix*}, *K whileAnnoFix-tr′*)]
end
⟫

syntax *-Call* :: *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*) (*CALL* -- [*1000*,*1000*] *21*)
-GuardedCall :: *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*) ($CALL_g$ -- [*1000*,*1000*] *21*)
-CallAss:: *′a* ⇒ *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*)
(- :== *CALL* -- [*30*,*1000*,*1000*] *21*)
-Proc :: *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*) (*PROC* -- *21*)
-ProcAss:: *′a* ⇒ *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*)
(- :== *PROC* -- [*30*,*1000*,*1000*] *21*)
-GuardedCallAss:: *′a* ⇒ *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*)
(- :== $CALL_g$ -- [*30*,*1000*,*1000*] *21*)
-DynCall :: *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*) (*DYNCALL* -- [*1000*,*1000*] *21*)
-GuardedDynCall :: *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*) ($DYNCALL_g$ -- [*1000*,*1000*] *21*)
-DynCallAss:: *′a* ⇒ *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*)
(- :== *DYNCALL* -- [*30*,*1000*,*1000*] *21*)
-GuardedDynCallAss:: *′a* ⇒ *′p* ⇒ *actuals* ⇒ ((*′a*,*string*,*′f*,*′e*) *com*)
(- :== $DYNCALL_g$ -- [*30*,*1000*,*1000*] *21*)

$-Call\text{-}ev :: 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(CALL_E$ ----- $[1000,1000,1000,1000,1000]\ 21)$
$-GuardedCall\text{-}ev :: 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(CALL_{Eg}$ ----- $[1000,1000,1000,1000,1000]\ 21)$
$-CallAss\text{-}ev:: 'a \Rightarrow 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(\text{-} :== CALL_E$ ----- $[30,1000,1000,1000,1000,1000]\ 21)$
$-Proc\text{-}ev :: 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(PROC_E$ ----- $21)$
$-ProcAss\text{-}ev:: 'a \Rightarrow 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(\text{-} :== PROC_E$ ----- $[30,1000,1000,1000,1000,1000]\ 21)$
$-GuardedCallAss\text{-}ev:: 'a \Rightarrow 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(\text{-} :== CALL_{Eg}$ ----- $[30,1000,1000,1000,1000,1000]\ 21)$
$-DynCall\text{-}ev :: 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(DYNCALL_E$ ----- $[1000,1000,1000,1000,1000]\ 21)$
$-GuardedDynCall\text{-}ev :: 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(DYNCALL_{eg}$ ----- $[1000,1000,1000,1000,1000]\ 21)$
$-DynCallAss\text{-}ev:: 'a \Rightarrow 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(\text{-} :== DYNCALL$ ----- $[30,1000,1000,1000,1000,1000]\ 21)$
$-GuardedDynCallAss\text{-}ev:: 'a \Rightarrow 'p \Rightarrow actuals \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow 'e\ option \Rightarrow (('a,string,'f,'e)\ com)$
$(\text{-} :== DYNCALL_g$ ----- $[30,1000,1000,1000,1000,1000]\ 21)$

$-Bind:: ['s \Rightarrow 'v,\ idt,\ 'v \Rightarrow ('s,'p,'f,'e)\ com] \Rightarrow ('s,'p,'f,'e)\ com$
$(\text{-} \gg \text{-./ -}\ [22,1000,21]\ 21)$
$-bseq::('s,'p,'f,'e)\ com \Rightarrow ('s,'p,'f,'e)\ com \Rightarrow ('s,'p,'f,'e)\ com$
$(\text{-}\gg/\ \text{-}\ [22,\ 21]\ 21)$

$-FCall :: ['p,actuals,idt,(('a,string,'f,'e)\ com)] \Rightarrow (('a,string,'f,'e)\ com)$
$(CALL$ -- $\gg$ -./ - $[1000,1000,1000,21]\ 21)$
$-FCall\text{-}ev :: ['p,actuals,'e\ option,'e\ option,'e\ option,idt,(('a,string,'f,'e)\ com)] \Rightarrow (('a,string,'f,'e)\ com)$
$(CALL_e$ -- ---$\gg$ -./ - $[1000,1000,1000,1000,1000,1000,21]\ 21)$

translations
$-Bind\ e\ i\ c == CONST\ bind\ (\text{-}quote\ e)\ (\lambda i.\ c)$

-FCall p acts i c == -FCall p acts (λ*i. c*)
-bseq c d == CONST bseq c d

definition *Let'*:: [*'a*, *'a* => *'b*] => *'b*
  where *Let'* = *Let*

ML-file *hoare-syntax.ML*

parse-translation ⟪
  let val ev1 = (*Syntax.const* @{*const-syntax None*});
      val ev2 = (*Syntax.const* @{*const-syntax None*});
      val ev3 = (*Syntax.const* @{*const-syntax None*}) *in*
  [(@{*syntax-const -Call*}, *Hoare-Syntax.call-tr false false ev1 ev2 ev3*),
  (@{*syntax-const -FCall*}, *Hoare-Syntax.fcall-tr ev1 ev2 ev3*),
  (@{*syntax-const -CallAss*}, *Hoare-Syntax.call-ass-tr false false ev1 ev2 ev3*),
  (@{*syntax-const -GuardedCall*}, *Hoare-Syntax.call-tr false true ev1 ev2 ev3*),
  (@{*syntax-const -GuardedCallAss*}, *Hoare-Syntax.call-ass-tr false true ev1 ev2 ev3*),
  (@{*syntax-const -Proc*}, *Hoare-Syntax.proc-tr ev1 ev2 ev3*),
  (@{*syntax-const -ProcAss*}, *Hoare-Syntax.proc-ass-tr ev1 ev2 ev3*),
  (@{*syntax-const -DynCall*}, *Hoare-Syntax.call-tr true false ev1 ev2 ev3*),
  (@{*syntax-const -DynCallAss*}, *Hoare-Syntax.call-ass-tr true false ev1 ev2 ev3*),
  (@{*syntax-const -GuardedDynCall*}, *Hoare-Syntax.call-tr true true ev1 ev2 ev3*),
  (@{*syntax-const -GuardedDynCallAss*}, *Hoare-Syntax.call-ass-tr true true ev1 ev2 ev3*),
  (@{*syntax-const -Call-ev*}, *Hoare-Syntax.call-ev-tr false false*),
  (@{*syntax-const -FCall-ev*}, *Hoare-Syntax.fcall-ev-tr*),
  (@{*syntax-const -CallAss-ev*}, *Hoare-Syntax.call-ass-ev-tr false false*),
  (@{*syntax-const -GuardedCall-ev*}, *Hoare-Syntax.call-ev-tr false true*),
  (@{*syntax-const -GuardedCallAss-ev*}, *Hoare-Syntax.call-ass-ev-tr false true*),
  (@{*syntax-const -Proc-ev*}, *Hoare-Syntax.proc-ev-tr*),
  (@{*syntax-const -ProcAss-ev*}, *Hoare-Syntax.proc-ass-ev-tr*),
  (@{*syntax-const -DynCall-ev*}, *Hoare-Syntax.call-ev-tr true false*),
  (@{*syntax-const -DynCallAss-ev*}, *Hoare-Syntax.call-ass-ev-tr true false*),
  (@{*syntax-const -GuardedDynCall-ev*}, *Hoare-Syntax.call-ev-tr true true*),
  (@{*syntax-const -GuardedDynCallAss-ev*}, *Hoare-Syntax.call-ass-ev-tr true true*)]

end
⟩⟩

parse-translation ⟨⟨
[(@{*syntax-const -Assign*}, *Hoare-Syntax.assign-tr*),
(@{*syntax-const -Assign-ev*}, *Hoare-Syntax.assign-ev-tr*),
(@{*syntax-const -raise*}, *Hoare-Syntax.raise-tr*),
(@{*syntax-const -raise-ev*}, *Hoare-Syntax.raise-ev-tr*),
(@{*syntax-const -New*}, *Hoare-Syntax.new-tr*),
(@{*syntax-const -New-ev*}, *Hoare-Syntax.new-ev-tr*),
(@{*syntax-const -NNew*}, *Hoare-Syntax.nnew-tr*),
(@{*syntax-const -NNew-ev*}, *Hoare-Syntax.nnew-ev-tr*),
(@{*syntax-const -GuardedAssign*}, *Hoare-Syntax.guarded-Assign-tr*),
(@{*syntax-const -GuardedAssign-ev*}, *Hoare-Syntax.guarded-Assign-ev-tr*),
(@{*syntax-const -GuardedNew*}, *Hoare-Syntax.guarded-New-tr*),
(@{*syntax-const -GuardedNNew*}, *Hoare-Syntax.guarded-NNew-tr*),
(@{*syntax-const -GuardedNew-ev*}, *Hoare-Syntax.guarded-New-ev-tr*),
(@{*syntax-const -GuardedNNew-ev*}, *Hoare-Syntax.guarded-NNew-ev-tr*),
(@{*syntax-const -GuardedWhile-inv-var*}, *Hoare-Syntax.guarded-While-tr*),
(@{*syntax-const -GuardedWhileFix-inv-var-hook*}, *Hoare-Syntax.guarded-WhileFix-tr*),
(@{*syntax-const -GuardedCond*}, *Hoare-Syntax.guarded-Cond-tr*),
(@{*syntax-const -GuardedAwait*}, *Hoare-Syntax.guarded-Await-tr*),
(@{*syntax-const -GuardedAwait-ev*}, *Hoare-Syntax.guarded-Await-ev-tr*),
(@{*syntax-const -Basic*}, *Hoare-Syntax.basic-tr*),
(@{*syntax-const -Basic-ev*}, *Hoare-Syntax.basic-ev-tr*)]
⟩⟩

parse-translation ⟨⟨
[(@{*syntax-const -Init*}, *Hoare-Syntax.init-tr*),
(∗ (@{*syntax-const -Init-ev*}, *Hoare-Syntax.init-ev-tr*), ∗)
(@{*syntax-const -Loc*}, *Hoare-Syntax.loc-tr*)]
⟩⟩

print-translation ⟨⟨
[(@{*const-syntax Basic*}, *Hoare-Syntax.assign-tr′*),
(@{*const-syntax raise*}, *Hoare-Syntax.raise-tr′*),
(@{*const-syntax Basic*}, *Hoare-Syntax.new-tr′*),
(@{*const-syntax Basic*}, *Hoare-Syntax.init-tr′*),
(@{*const-syntax Spec*}, *Hoare-Syntax.nnew-tr′*),
(@{*const-syntax block*}, *Hoare-Syntax.loc-tr′*),
(@{*const-syntax Collect*}, *Hoare-Syntax.assert-tr′*),
(@{*const-syntax Cond*}, *Hoare-Syntax.bexp-tr′ -Cond*),
(@{*const-syntax switch*}, *Hoare-Syntax.switch-tr′*),
(@{*const-syntax Basic*}, *Hoare-Syntax.basic-tr′*),
(@{*const-syntax guards*}, *Hoare-Syntax.guards-tr′*),

(@{*const-syntax whileAnnoG*}, *Hoare-Syntax.whileAnnoG-tr′*),
(@{*const-syntax whileAnnoGFix*}, *Hoare-Syntax.whileAnnoGFix-tr′*),
(@{*const-syntax bind*}, *Hoare-Syntax.bind-tr′*)]
⟫

print-translation ⟪
```
  let
    fun spec-tr' ctxt ((coll as Const -)$
                 ((splt as Const -) $ (t as (Abs (s,T,p))))::ts) =
        let
          fun selector (Const (c, T)) = Hoare.is-state-var c
            | selector (Const (@{syntax-const -free}, -) $ (Free (c, T))) =
                Hoare.is-state-var c
            | selector - = false;
        in
          if Hoare-Syntax.antiquote-applied-only-to selector p then
            Syntax.const @{const-syntax Spec} $ coll $
              (splt $ Hoare-Syntax.quote-mult-tr' ctxt selector
                      Hoare-Syntax.antiquoteCur Hoare-Syntax.antiquoteOld  (Abs
(s,T,t)))
           else raise Match
        end
      | spec-tr' - ts = raise Match
  in [(@{const-syntax Spec}, spec-tr')] end
```
⟫

print-translation ⟪
[(@{*const-syntax call*}, *Hoare-Syntax.call-tr′*),
(@{*const-syntax dynCall*}, *Hoare-Syntax.dyn-call-tr′*),
(@{*const-syntax fcall*}, *Hoare-Syntax.fcall-tr′*),
(@{*const-syntax Call*}, *Hoare-Syntax.proc-tr′*)]
⟫

nonterminal *prgs*

syntax
-PAR :: *prgs* ⇒ *′a* (*COBEGIN*//-//*COEND 60*)
-prg :: *′a* ⇒ *prgs* (- *57*)
-prgs :: [*′a*, *prgs*] ⇒ *prgs* (-//‖//- [*60*,*57*] *57*)

translations
-prg a ⇀ [*a*]
-prgs a ps ⇀ *a* # *ps*
-PAR ps ⇀ *ps*

syntax

-prg-scheme :: [*'a*, *'a*, *'a*, *'a*] ⇒ *prgs* (*SCHEME* [- ≤ - < -] - [*0*,*0*,*0*,*60*] *57*)

translations
-prg-scheme j i k c ⇌ (*CONST map* (λ*i*. *c*) [*j*..<*k*])

Translations for variables before and after a transition:

syntax
-before :: *id* ⇒ *'a* (º-)
-after :: *id* ⇒ *'a* (ª-)

translations
º*x* == *x* ´ *CONST fst*
ª*x* == *x* ´ *CONST snd*

end

theory *XVcgCon*
imports *VcgCon*

begin

We introduce a syntactic variant of the let-expression so that we can safely unfold it during verification condition generation. With the new theorem attribute *vcg-simp* we can declare equalities to be used by the verification condition generator, while simplifying assertions.

syntax
-Let' :: [*letbinds*, *basicblock*] => *basicblock* ((*LET* (-)/ *IN* (-)) *23*)

translations
-Let' (*-binds b bs*) *e* == *-Let' b* (*-Let' bs e*)
-Let' (*-bind x a*) *e* == *CONST Let' a* (%*x*. *e*)

lemma *Let'-unfold* [*vcg-simp*]: *Let' x f* = *f x*
by (*simp add*: *Let'-def Let-def*)

lemma *Let'-split-conv* [*vcg-simp*]:
(*Let' x* (λ*p*. (*case-prod* (*f p*) (*g p*)))) =
(*Let' x* (λ*p*. (*f p*) (*fst* (*g p*)) (*snd* (*g p*))))
by (*simp add*: *split-def*)

end