Abstract

We present the theory of Simpl, a sequential imperative programming language. We introduce its syntax, its semantics (big and small-step operational semantics) and Hoare logics for both partial as well as total correctness. We prove soundness and completeness of the Hoare logic. We integrate and automate the Hoare logic in Isabelle/HOL to obtain a practically usable verification environment for imperative programs.

Simpl is independent of a concrete programming language but expressive enough to cover all common language features: mutually recursive procedures, abrupt termination and exceptions, runtime faults, local and global variables, pointers and heap, expressions with side effects, pointers to procedures, partial application and closures, dynamic method invocation and also unbounded nondeterminism.

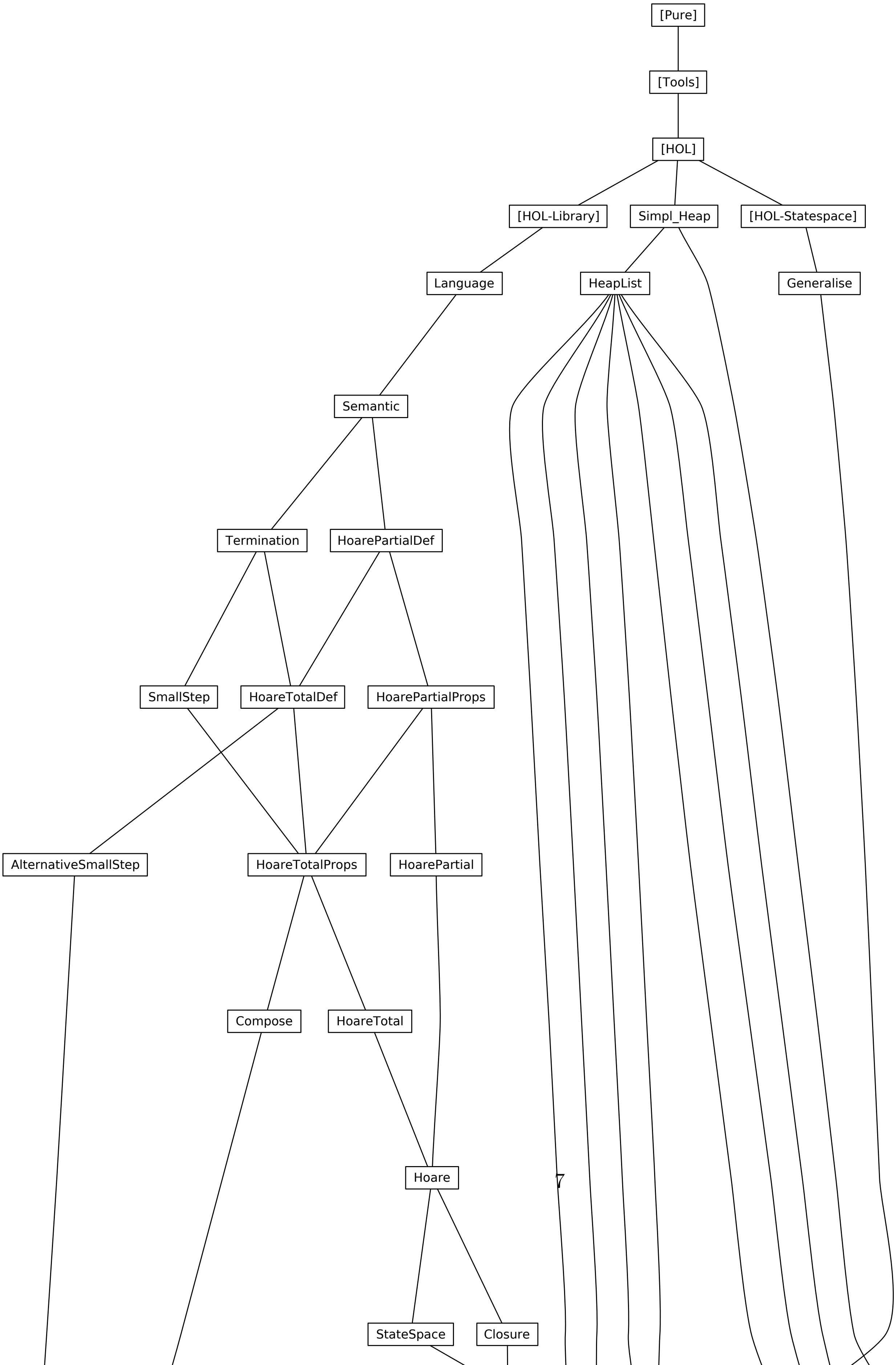
— Simpl —

A Sequential Imperative Programming Language Syntax, Semantics, Hoare Logics and Verification Environment

Norbert W. Schirmer

April 20, 2020

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
Simpl_Heap
[HOL-Statespace]
Language
HeapList
Generalise
Semantic
Termination
HoarePartialDef
SmallStep
HoareTotalDef
HoarePartialProps
AlternativeSmallStep
HoareTotalProps
HoarePartial
Compose
HoareTotal
Hoare
StateSpace
Closure

1 Introduction

The work presented in these theories was developed within the German Verisoft project[1]. A thorough description of the core parts can be found in my PhD thesis [9]. A tutorial-like user guide is in Section 26.

Applications so far include BDD-normalisation [8], a C0 compiler [4], a page fault handler [1] and extensions towards separation logic [10].

2 The Simpl Syntax

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

2.1 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* =
  Skip
  | *Basic 's* ⇒ *'s*
  | *Spec* (*'s* × *'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* ⇒ (*'s,'p,'f*) *com*
  | *Guard 'f 's bexp* (*'s,'p,'f*) *com*
  | *Throw*
  | *Catch* (*'s,'p,'f*) *com* (*'s,'p,'f*) *com*

2.2 Derived Language Constructs

definition
  raise:: (*'s* ⇒ *'s*) ⇒ (*'s,'p,'f*) *com* **where**
  raise f = *Seq* (*Basic f*) *Throw*

definition
  condCatch:: (*'s,'p,'f*) *com* ⇒ *'s bexp* ⇒ (*'s,'p,'f*) *com* ⇒ (*'s,'p,'f*) *com* **where**
  condCatch c_1 *b* c_2 = *Catch* c_1 (*Cond b* c_2 *Throw*)

definition
  bind:: (*'s* ⇒ *'v*) ⇒ (*'v* ⇒ (*'s,'p,'f*) *com*) ⇒ (*'s,'p,'f*) *com* **where**
  bind e c = *DynCom* (λ*s. c* (*e s*))

[1] http://www.verisoft.de

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

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*)

2.3 Operations on Simpl-Syntax

2.3.1 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* ($\lambda s.$ (*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)

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* (λv. *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* ($\lambda s\ t$. *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* ($\lambda i\ t$. *normalize* (*c i t*))
by (*simp add*: *call-def*)

lemma *normalize-dynCall* [*simp*]:
normalize (*dynCall init p return c*) =
dynCall init p return ($\lambda s\ t$. *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 (λv. *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

2.3.2 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* ($\lambda s.$ (*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 = ($\lambda p.$ *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* ($\lambda v.$ *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* (λ*s 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 (λ*s 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 (λ*s 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 (λ*v*. *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* ($\lambda(f,g)$. $f \notin F$) *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

2.3.3 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* (λs. (*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* (λv. *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* (λs *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 (λs *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 (λs *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* (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) *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*

2.3.4 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* ($\lambda s.$ (*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* $c' \Longrightarrow$
$\exists c''. c = DynCom\ c'' \wedge (\lambda s. (merge\text{-}guards\ (c''\ s))) = c'$
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* $\Longrightarrow$
$\exists c1'\ c2'. c = Catch\ c1'\ c2' \wedge merge\text{-}guards\ c1' = c1 \wedge merge\text{-}guards\ c2' = c2$
by (*cases c*) (*auto split*: *com.splits if-split-asm simp add*: *is-Guard-def Let-def*)

lemma *merge-guards-res-Guard*:
$merge\text{-}guards\ c = Guard\ f\ g\ c' \Longrightarrow \exists c''\ f'\ g'. c = Guard\ f'\ g'\ c''$
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* (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 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 $(\lambda s\ t.\ \textit{merge-guards}\ (c\ s\ t))$
by (*simp add*: *call-def*)

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

lemma *merge-guards-fcall* [*simp*]:
merge-guards (*fcall init p return result c*) =
fcall init p return result $(\lambda v.\ \textit{merge-guards}\ (c\ v))$
by (*simp add*: *fcall-def*)

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

lemma *merge-guards-guaranteeStrip* [*simp*]:
merge-guards (*guaranteeStrip f g c*) =
$(\textit{let}\ c' = (\textit{merge-guards}\ c)$
$\textit{in if is-Guard}\ c'$
$\textit{then let}\ (f',g',c') = \textit{dest-Guard}\ c'$
$\textit{in if}\ f{=}f'\ \textit{then Guard}\ f\ (g \cap g')\ c'$
$\textit{else Guard}\ f\ g\ (\textit{Guard}\ f'\ g'\ c')$
$\textit{else Guard}\ f\ g\ c')$
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 $(\lambda s.\ \textit{merge-guards}\ (c\ \textit{undefined}))$ *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*:: $('s,'p,'f)\ \textit{com} \Rightarrow \textit{bool}$
where
noguards Skip = *True* |
noguards (*Basic f*) = *True* |

noguards (*Spec* r) = *True* |
noguards (*Seq* c_1 c_2) = (*noguards* c_1 ∧ *noguards* c_2) |
noguards (*Cond* b c_1 c_2) = (*noguards* c_1 ∧ *noguards* c_2) |
noguards (*While* b c) = (*noguards* c) |
noguards (*Call* p) = *True* |
noguards (*DynCom* c) = (∀ s. *noguards* (c s)) |
noguards (*Guard* f g c) = *False* |
noguards *Throw* = *True* |
noguards (*Catch* c_1 c_2) = (*noguards* c_1 ∧ *noguards* c_2)

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

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

2.3.5 Intersecting Guards: $c_1 \cap_g c_2$

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

inductive-cases *com-rel-elim-cases*:
(*c*, *Skip*) ∈ *com-rel*
(*c*, *Basic f*) ∈ *com-rel*
(*c*, *Spec r*) ∈ *com-rel*
(*c*, *Seq c1 c2*) ∈ *com-rel*
(*c*, *Cond b c1 c2*) ∈ *com-rel*
(*c*, *While b c1*) ∈ *com-rel*
(*c*, *Call p*) ∈ *com-rel*

$(c,\ DynCom\ c1) \in com\text{-}rel$
$(c,\ Guard\ f\ g\ c1) \in com\text{-}rel$
$(c,\ Throw) \in com\text{-}rel$
$(c,\ Catch\ c1\ c2) \in com\text{-}rel$

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 $c \cap_g d == inter\text{-}guards\ (c,d)$

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 ∩g c2 of
          None ⇒ None
        | Some c ⇒ Some (While cnd1 c))
     else None)
  (Call p1 ∩g Call p2) =
    (if p1 = p2
     then Some (Call p1)
     else None)
  (DynCom P1 ∩g DynCom P2) =
    (if (∀s. (P1 s ∩g P2 s) ≠ None)
    then Some (DynCom (λs. the (P1 s ∩g P2 s)))
    else None)
  (Guard m1 g1 c1 ∩g Guard m2 g2 c2) =
    (if m1 = m2 then
      (case c1 ∩g c2 of
         None ⇒ None
       | Some c ⇒ Some (Guard m1 (g1 ∩ g2) c))
     else None)
  (Throw ∩g Throw) = Some Throw
  (Catch a1 a2 ∩g Catch b1 b2) =
    (case a1 ∩g b1 of
       None ⇒ None
     | Some c1 ⇒ (case a2 ∩g b2 of
         None ⇒ None
       | Some c2 ⇒ Some (Catch c1 c2)))
  (c ∩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:
  ⋀c. (c1 ∩g c2) = Some c  ⟹
    (strip-guards UNIV c = strip-guards UNIV c1) ∧
    (strip-guards UNIV c = strip-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: ⋀c. (c1 ∩g c2) = Some c ⟹ (c2 ∩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*

2.3.6 Subset on Guards: $c_1 \subseteq_g c_2$

inductive *subseteq-guards* :: $('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com \Rightarrow bool$
$(\text{-} \subseteq_g \text{-}\ [20,20]\ 19)$ **where**
$Skip \subseteq_g Skip$
$|\ f1 = f2 \Longrightarrow Basic\ f1 \subseteq_g Basic\ f2$
$|\ r1 = r2 \Longrightarrow Spec\ r1 \subseteq_g Spec\ r2$
$|\ a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow Seq\ a1\ a2 \subseteq_g Seq\ b1\ b2$
$|\ cnd1 = cnd2 \Longrightarrow t1 \subseteq_g t2 \Longrightarrow e1 \subseteq_g e2 \Longrightarrow Cond\ cnd1\ t1\ e1 \subseteq_g Cond\ cnd2\ t2\ e2$
$|\ cnd1 = cnd2 \Longrightarrow c1 \subseteq_g c2 \Longrightarrow While\ cnd1\ c1 \subseteq_g While\ cnd2\ c2$
$|\ p1 = p2 \Longrightarrow Call\ p1 \subseteq_g Call\ p2$
$|\ (\bigwedge s.\ P1\ s \subseteq_g P2\ s) \Longrightarrow DynCom\ P1 \subseteq_g DynCom\ P2$
$|\ m1 = m2 \Longrightarrow g1 = g2 \Longrightarrow c1 \subseteq_g c2 \Longrightarrow Guard\ m1\ g1\ c1 \subseteq_g Guard\ m2\ g2\ c2$
$|\ c1 \subseteq_g c2 \Longrightarrow c1 \subseteq_g Guard\ m2\ g2\ c2$
$|\ Throw \subseteq_g Throw$
$|\ a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow Catch\ a1\ a2 \subseteq_g Catch\ b1\ b2$

lemma *subseteq-guards-Skip*:
$c = Skip$ **if** $c \subseteq_g Skip$
using *that* **by** *cases*

lemma *subseteq-guards-Basic*:

$c = Basic\ f$ **if** $c \subseteq_g Basic\ f$
using *that* **by** *cases simp*

lemma *subseteq-guards-Spec*:
$c = Spec\ r$ **if** $c \subseteq_g Spec\ r$
using *that* **by** *cases simp*

lemma *subseteq-guards-Seq*:
$\exists c1'\ c2'.\ c = Seq\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$ **if** $c \subseteq_g Seq\ c1\ c2$
using *that* **by** *cases simp*

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

lemma *subseteq-guards-While*:
$\exists c''.\ c{=}While\ b\ c'' \wedge (c'' \subseteq_g c')$ **if** $c \subseteq_g While\ b\ c'$
using *that* **by** *cases simp*

lemma *subseteq-guards-Call*:
$c = Call\ p$ **if** $c \subseteq_g Call\ p$
using *that* **by** *cases simp*

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

lemma *subseteq-guards-Guard*:
$(c \subseteq_g c') \vee (\exists c''.\ c = Guard\ f\ g\ c'' \wedge (c'' \subseteq_g c'))$ **if** $c \subseteq_g Guard\ f\ g\ c'$
using *that* **by** *cases simp-all*

lemma *subseteq-guards-Throw*:
$c = Throw$ **if** $c \subseteq_g Throw$
using *that* **by** *cases*

lemma *subseteq-guards-Catch*:
$\exists c1'\ c2'.\ c = Catch\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$ **if** $c \subseteq_g Catch\ c1\ c2$
using *that* **by** *cases simp*

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'*:
$\exists f'\ b'\ c'.\ d = Guard\ f'\ b'\ c'$ **if** $Guard\ f\ b\ c \subseteq_g d$
using *that* **by** *cases auto*

lemma *subseteq-guards-refl*: $c \subseteq_g c$

by (*induct c*) (*auto intro*: *subseteq-guards.intros*)

end

3 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*:
¬ *isAbr s* ⟹ (∃ *s'*. *s*=*Normal s'*) ∨ *s* = *Stuck* ∨ (∃ *f*. *s*=*Fault f*)
by (*cases s*) *auto*

definition *isFault*:: (*'s*,*'f*) *xstate* ⇒ *bool*
where *isFault S* = (∃ *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*: (¬ *isFault t*) = (∀ *f*. *t* ≠ *Fault f*)
by (*auto elim*: *isFaultE*)

3.1 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
$exec::[('s,'p,'f)\ body,('s,'p,'f)\ com,('s,'f)\ xstate,('s,'f)\ xstate]$
$\Rightarrow bool\ (\text{-}\vdash \langle\text{-},\text{-}\rangle \Rightarrow \text{-}\ \ [60,20,98,98]\ 89)$
for $\Gamma::('s,'p,'f)\ body$
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*:
Γ 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* (Γ 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*:
$\bigwedge bdy\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma \vdash \langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t';$
$\Gamma \vdash \langle c\ s\ t', Normal\ (return\ s\ t')\rangle \Rightarrow\ t]\!]$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge bdy\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma \vdash \langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Abrupt\ t';$
$t = Abrupt\ (return\ s\ t')]\!]$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge bdy\ f.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma \vdash \langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Fault\ f;$
$t = Fault\ f]\!]$
$\Longrightarrow P$
assumes *Stuck*:
$\bigwedge bdy.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma \vdash \langle bdy, Normal\ (init\ s)\rangle \Rightarrow\ Stuck;$
$t = Stuck]\!]$
$\Longrightarrow P$
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*:
$[\![\Gamma\vdash\langle \mathit{call\ init\ (p\ s)\ return\ c,Normal\ s}\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{dynCall\ init\ p\ return\ c,Normal\ s}\rangle \Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

lemma *exec-dynCall-Normal-elim*:
assumes *exec*: $\Gamma\vdash\langle \mathit{dynCall\ init\ p\ return\ c,Normal\ s}\rangle \Rightarrow\ t$
assumes *call*: $\Gamma\vdash\langle \mathit{call\ init\ (p\ s)\ return\ c,Normal\ s}\rangle \Rightarrow\ t \Longrightarrow P$
shows *P*
using *exec*

apply (*simp add*: *dynCall-def*)
apply (*erule exec-Normal-elim-cases*)
apply (*rule call*,*assumption*)
done

lemma *exec-Call-body*:
Γ *p*=*Some bdy* $\Longrightarrow$
$\Gamma\vdash\langle$*Call p*,*s*$\rangle \Rightarrow$ *t* = $\Gamma\vdash\langle$*the* (Γ *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'*)

3.2 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* (-$\vdash$ $\langle$-,-$\rangle$ =-$\Rightarrow$ - [*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*: $(s,t) \in r$
$\Longrightarrow$
$\Gamma\vdash\langle \mathit{Spec}\ r,\mathit{Normal}\ s\rangle =n\Rightarrow\ \mathit{Normal}\ t$

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

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

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

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

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

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

$\Gamma\vdash\langle Catch\ c_1\ \ c_2,Normal\ s\rangle\ =n\Rightarrow\ t$
| *CatchMiss*: $[\![\Gamma\vdash\langle c_1,Normal\ s\rangle\ =n\Rightarrow\ \ t;\ \neg isAbr\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle Catch\ c_1\ \ c_2,Normal\ s\rangle\ =n\Rightarrow\ \ t$

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

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

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*: Γ⊢⟨*c*,*s*⟩ =*n*⇒ *t*
shows Γ⊢⟨*c*,*s*⟩ ⇒ *t*
using *execn*
by *induct* (*auto intro*: *exec.intros*)

lemma *exec-to-execn*:
assumes *execn*: Γ⊢⟨*c*,*s*⟩ ⇒ *t*
shows ∃ *n*. Γ⊢⟨*c*,*s*⟩ =*n*⇒ *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**
Γ⊢⟨*c1*,*Normal s*⟩ =*n*⇒ *s′* Γ⊢⟨*c2*,*s′*⟩ =*m*⇒ *s′′*
by *blast*
then have
Γ⊢⟨*c1*,*Normal s*⟩ =*max n m*⇒ *s′*
Γ⊢⟨*c2*,*s′*⟩ =*max n m*⇒ *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**
Γ⊢⟨*c*,*Normal s*⟩ =*n*⇒ *s′* Γ⊢⟨*While b c*,*s′*⟩ =*m*⇒ *s′′*
by *blast*
then have
Γ⊢⟨*c*,*Normal s*⟩ =*max n m*⇒ *s′* Γ⊢⟨*While b c*,*s′*⟩ =*max n m*⇒ *s′′*

  by (*auto elim*!: *execn-mono intro*: *max.coboundedI 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*:: $('s,'p,'f)\ body \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'f)\ xstate \Rightarrow\ nat$
      $\Rightarrow ('s,'f)\ xstate\ set \Rightarrow bool$
 $(\text{-}\vdash \langle\text{-},\text{-}\rangle =\text{-}\Rightarrow\notin\text{-}\ \ [60,20,98,65,60]\ 89)$ **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*:: $('s,'p,'f)\ body \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'f)\ xstate$
      $\Rightarrow ('s,'f)\ xstate\ set \Rightarrow bool$
 $(\text{-}\vdash \langle\text{-},\text{-}\rangle \Rightarrow\notin\text{-}\ \ [60,20,98,60]\ 89)$ **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*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq \mathit{Fault}\ f]\!] \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{\mathit{Fault}\ f\}$
by (*simp add*: *final-notin-def*)

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

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

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

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

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

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

lemma *noStuck-def'*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{\mathit{Stuck}\} = (\neg\Gamma\vdash\langle c,s\rangle \Rightarrow \mathit{Stuck})$
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*:
$[\![\Gamma \vdash \langle c,s\rangle =n\Rightarrow \notin \{Fault\ True, Fault\ False, Stuck\};\ \Gamma \vdash \langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow$
$t \notin \{Fault\ True, Fault\ False, Stuck\}$
by (*simp add*: *nfinal-notin-def*)

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

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

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

lemma *noStuck-Call*:

assumes *noStuck*: $\Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
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 $\Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow Stuck$
by (*rule exec.CallUndefined*)
with *noStuck* **show** *?thesis*
by (*auto simp add*: *final-notin-def*)
qed

lemma *Guard-noFaultStuckD*:
assumes $\Gamma\vdash\langle Guard\ f\ g\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
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*: $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin T$
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*:
$\Gamma\ p=Some\ bdy\Longrightarrow$
$\Gamma\vdash\langle Call\ p\ , Normal\ s\rangle \Rightarrow\notin\{Fault\ f\} =$
$\Gamma\vdash\langle the\ (\Gamma\ p), Normal\ s\rangle \Rightarrow\notin\{Fault\ f\}$
by (*simp add*: *noFault-def' exec-Call-body*)

lemma *noStuck-Call-body*:
$\Gamma\ p=Some\ bdy\Longrightarrow$
$\Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin\{Stuck\} =$
$\Gamma\vdash\langle the\ (\Gamma\ p), Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
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 $t \in Fault\ `\ F$
      then obtain *f* **where**
      t: $t=Fault\ f$ **and** *f*: $f \in F$
        by *auto*
      from *t exec-c1*
      have $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow Fault\ f$
        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*: $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
  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'*)

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

lemma *execn-sequence-app*: $\bigwedge s\ s'\ t.$
$[\![\Gamma\vdash\langle sequence\ Seq\ xs,Normal\ s\rangle =n\Rightarrow s'; \Gamma\vdash\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow t]\!]$
$\Longrightarrow \Gamma\vdash\langle sequence\ Seq\ (xs@ys),Normal\ s\rangle =n\Rightarrow t$
proof (*induct xs*)
  case *Nil*
  thus *?case* **by** (*auto elim*: *execn-Normal-elim-cases*)
next
  case (*Cons x xs*)
  have *exec-x-xs*: $\Gamma\vdash\langle sequence\ Seq\ (x\ \#\ xs),Normal\ s\rangle =n\Rightarrow s'$ **by** *fact*
  have *exec-ys*: $\Gamma\vdash\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow t$ **by** *fact*
  show *?case*
  proof (*cases xs*)
    case *Nil*
    with *exec-x-xs* **have** $\Gamma\vdash\langle x,Normal\ s\rangle =n\Rightarrow s'$
      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: $\Gamma\vdash\langle x,Normal\ s\rangle =n\Rightarrow s''$ **and**
      exec-xs: $\Gamma\vdash\langle sequence\ Seq\ xs,s''\rangle =n\Rightarrow s'$
      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 $\Gamma\vdash\langle sequence\ Seq\ (xs\ @\ ys),Normal\ s'''\rangle =n\Rightarrow t$ **.**
      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*: $\bigwedge s\ t.\ \Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),*Normal s*$\rangle$ $=n\Rightarrow$ t $\Longrightarrow$
$\exists s'.\ \Gamma\vdash\langle$*sequence Seq xs*,*Normal s*$\rangle$ $=n\Rightarrow$ $s' \wedge \Gamma\vdash\langle$*sequence Seq ys*,$s'\rangle$ $=n\Rightarrow$ t
proof (*induct xs*)
case *Nil*
thus *?case*
by (*auto intro*: *execn.intros*)
next
case (*Cons x xs*)
have *exec-app*: $\Gamma\vdash\langle$*sequence Seq* ((*x* # *xs*) @ *ys*),*Normal s*$\rangle$ $=n\Rightarrow$ 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: $\Gamma\vdash\langle x$,*Normal s*$\rangle$ $=n\Rightarrow$ s' **and**
exec-xs-ys: $\Gamma\vdash\langle$*sequence Seq* (*xs* @ *ys*),$s'\rangle$ $=n\Rightarrow$ 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*]:
  $[\![\Gamma\vdash\langle sequence\ Seq\ (xs\ @\ ys),Normal\ s\rangle =n\Rightarrow\ t;$
   $\bigwedge s'.\ [\![\Gamma\vdash\langle sequence\ Seq\ xs,Normal\ s\rangle =n\Rightarrow\ s';\Gamma\vdash\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow\ t]\!]$
$\Longrightarrow P$
   $]\!] \Longrightarrow P$
  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.intros 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*⇒ *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*⇒ *s''* **and**
$\Gamma\vdash\langle$*sequence Seq* (*flatten c2*),*s''*$\rangle$ =*n*⇒ *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*⇒ *t* ⟹ $\Gamma\vdash\langle$*c*,*s*$\rangle$ =*n*⇒ *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*⇒ *t* **by** *fact*
hence *exec-norm-seq*:
$\Gamma\vdash\langle$*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*)),*s*$\rangle$ =*n*⇒ *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*⇒ *s''*
and
exec-norm-c2: $\Gamma\vdash\langle$*sequence Seq* (*flatten* (*normalize c2*)),*s''*$\rangle$ =*n*⇒ *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: Γ⊢⟨*sequence Seq xs*,*Normal s*⟩ =*n*⇒ *s'*..
from *exec-to-execn* [*OF exec-ys*]
obtain *m* **where**
execn-ys: Γ⊢⟨*sequence Seq ys*,*s'*⟩ =*m*⇒ *t*..
with *execn-xs* **obtain**
Γ⊢⟨*sequence Seq xs*,*Normal s*⟩ =*max n m*⇒ *s'*
Γ⊢⟨*sequence Seq ys*,*s'*⟩ =*max n m*⇒ *t*
by (*auto intro*: *execn-mono max.cobounded1 max.cobounded2*)
from *execn-sequence-app* [*OF this*]
have Γ⊢⟨*sequence Seq* (*xs* @ *ys*),*Normal s*⟩ =*max n m*⇒ *t* **.**
thus *?thesis*
by (*rule execn-to-exec*)
qed

lemma *exec-sequence-appD*:
assumes *exec-xs-ys*: Γ⊢⟨*sequence Seq* (*xs* @ *ys*),*Normal s*⟩ ⇒ *t*
shows ∃ *s'*. Γ⊢⟨*sequence Seq xs*,*Normal s*⟩ ⇒ *s'* ∧ Γ⊢⟨*sequence Seq ys*,*s'*⟩ ⇒ *t*
proof −
from *exec-to-execn* [*OF exec-xs-ys*]
obtain *n* **where** Γ⊢⟨*sequence Seq* (*xs* @ *ys*),*Normal s*⟩ =*n*⇒ *t*..

thus *?thesis*
by (*cases rule*: *execn-sequence-appE*) (*auto intro*: *execn-to-exec*)
qed

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

3.4 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 $\Gamma\vdash\langle c2,Normal\ s'\rangle =n\Rightarrow t$
with *c2-c2' Normal Cond.hyps* **obtain** t' **where**
$\Gamma\vdash\langle c2',Normal\ s'\rangle =n\Rightarrow t'$
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*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$ **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 *c r w*
assume *exec*: $\Gamma\vdash\langle c,r\rangle =n\Rightarrow w$
assume *c*: $c=$*While b c''*
have $\exists w'.\ \Gamma\vdash\langle While\ b\ c',r\rangle =n\Rightarrow w' \wedge$
(*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''*: $\Gamma\vdash\langle c'',Normal\ r\rangle =n\Rightarrow u$ **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: Γ⊢⟨c,s⟩ =n⇒ t by fact
  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
  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 Γ⊢⟨C s',Normal s'⟩ =n⇒ t
      by cases
    from DynCom.hyps C-C' [rule-format] this obtain t' where
      Γ⊢⟨C' s',Normal s'⟩ =n⇒ t'
      isFault t ⟶ isFault t'
      ¬ isFault t' ⟶ 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*: $(c \subseteq_g c') \vee (\exists c''.\ c = \mathit{Guard}\ f'\ g'\ c'' \wedge (c'' \subseteq_g c'))$
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': $\Gamma\vdash\langle c',\mathit{Normal}\ s'\rangle =n\Rightarrow t'$ **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 $\exists c''.\ c = \mathit{Guard}\ f'\ g'\ c'' \wedge (c'' \subseteq_g c')$
then obtain c'' **where**
c: $c = \mathit{Guard}\ f'\ g'\ c''$ **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 $c \subseteq_g$ *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*
have *exec*: $\Gamma\vdash\langle c,s\rangle$ =*n*$\Rightarrow$ *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
        Γ⊢⟨c2',Normal w⟩ =n⇒ t'
        isFault t ⟶ isFault t'
        ¬ isFault t' ⟶ t' = t
        by blast
      with exec-c1' w' Normal
      show ?thesis
        by (fastforce intro: execn.intros )
    qed
  next
    assume exec-c1: Γ⊢⟨c1,Normal s'⟩ =n⇒ t
    assume t: ¬ isAbr t
    from Normal exec-c1 c1-c1' Catch.hyps obtain t' where
      exec-c1': Γ⊢⟨c1',Normal s'⟩ =n⇒ 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' 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 $\exists\, t'.\ \Gamma\vdash\langle c',s\rangle \Rightarrow t' \land$
$(\mathit{isFault}\ t \longrightarrow \mathit{isFault}\ t') \land (\neg\ \mathit{isFault}\ t' \longrightarrow t'=t)$
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

3.5 Lemmas about *merge-guards*

theorem *execn-to-execn-merge-guards*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
shows $\Gamma\vdash\langle \mathit{merge\text{-}guards}\ c,s\rangle =n\Rightarrow t$
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*: $\Gamma\vdash\langle \mathit{merge\text{-}guards}\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$ **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* $\exists f'\ g'\ c'.$ *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*:
$\bigwedge s\ n\ t.\ \Gamma\vdash\langle \textit{merge-guards } c,\textit{Normal } s\rangle =n\Rightarrow t \Longrightarrow \Gamma\vdash\langle c,\textit{Normal } s\rangle =n\Rightarrow t$
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 $\Gamma\vdash\langle \textit{merge-guards } (\textit{Seq c1 c2}),\textit{Normal } s\rangle =n\Rightarrow t$ **by** *fact*
hence *exec-merge*: $\Gamma\vdash\langle \textit{Seq } (\textit{merge-guards c1}) (\textit{merge-guards c2}),\textit{Normal } s\rangle =n\Rightarrow t$
by *simp*
then obtain s' **where**
exec-merge-c1: $\Gamma\vdash\langle \textit{merge-guards c1},\textit{Normal } s\rangle =n\Rightarrow s'$ **and**
exec-merge-c2: $\Gamma\vdash\langle \textit{merge-guards c2},s'\rangle =n\Rightarrow t$
by *cases*

```
  from exec-merge-c1
  have exec-c1: Γ⊢⟨c1,Normal s⟩ =n⇒ s'
    by (rule Seq.hyps)
  show ?case
  proof (cases s')
    case (Normal s'')
    with exec-merge-c2
    have Γ⊢⟨c2,s'⟩ =n⇒ 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* ($g \cap g'$) c'
by *simp*
show *?thesis*
proof (*cases* $s \in g'$)
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*:
  $\Gamma\vdash\langle$*merge-guards c*,*s*$\rangle$ =*n*$\Rightarrow$ *t* $\Longrightarrow$ $\Gamma\vdash\langle$*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*:
 $\Gamma\vdash\langle$*c*, *s*$\rangle$ =*n*$\Rightarrow$ *t* = $\Gamma\vdash\langle$*merge-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*:
 $\Gamma\vdash\langle$*c*, *s*$\rangle$ $\Rightarrow$ *t* = $\Gamma\vdash\langle$*merge-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*:
 $\Gamma\vdash\langle$*c*, *s*$\rangle$ $\Rightarrow$ *t* $\Longrightarrow$ $\Gamma\vdash\langle$*merge-guards c*,*s*$\rangle$ $\Rightarrow$ *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*])

3.6 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*: 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*
  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**
      Γ⊢⟨*mark-guards x c1*,*Normal s*⟩ =*n*⇒ *Fault f''*
      by *auto*
    moreover have Γ⊢⟨*mark-guards x c2*,*Fault f''*⟩ =*n*⇒ *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*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ w$ **by** *fact*
  have *exec-w*: $\Gamma\vdash\langle While\ b\ c,w\rangle =n\Rightarrow\ t$ **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**
      $\Gamma\vdash\langle mark\text{-}guards\ x\ c,Normal\ s\rangle =n\Rightarrow\ Fault\ f''$
      by *auto*
    moreover have $\Gamma\vdash\langle mark\text{-}guards\ x\ (While\ b\ c),Fault\ f''\rangle =n\Rightarrow\ Fault\ f''$
      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*: $\Gamma\vdash\langle mark\text{-}guards\ x\ c,Normal\ s\rangle =n\Rightarrow\ w$
      by *simp*
    with *WhileTrue.hyps t* **obtain** f' **where**
      $\Gamma\vdash\langle mark\text{-}guards\ x\ (While\ b\ c),w\rangle =n\Rightarrow\ Fault\ f'$
      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*: $\Gamma\vdash\langle mark\text{-}guards\ x\ c,Normal\ s\rangle =n\Rightarrow\ w$
      by *simp*
    with *WhileTrue.hyps t* **obtain** f' **where**
      $\Gamma\vdash\langle mark\text{-}guards\ x\ (While\ b\ c),w\rangle =n\Rightarrow\ Fault\ f'$
      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*: $\Gamma\vdash\langle mark\text{-}guards\ x\ c1,Normal\ s\rangle =n\Rightarrow Abrupt\ w$
by *simp*
with *CatchMatch.hyps t* **obtain** f' **where**
$\Gamma\vdash\langle mark\text{-}guards\ x\ c2,Normal\ w\rangle =n\Rightarrow Fault\ f'$
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*:
$\bigwedge s\ n\ t.\ \Gamma\vdash\langle mark\text{-}guards\ f\ c,s\rangle =n\Rightarrow t$
$\Longrightarrow \exists t'.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t' \wedge$
$(isFault\ t \longrightarrow isFault\ t') \wedge$
$(t' = Fault\ f \longrightarrow t'=t) \wedge$
$(isFault\ t' \longrightarrow isFault\ t) \wedge$
$(\neg\ isFault\ t' \longrightarrow t'=t)$
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: Γ⊢⟨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
          Γ⊢⟨c2,w⟩ =n⇒ t' and
          isFault t ⟶ isFault t' and
          t' = Fault f ⟶ t'=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 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* $s' \in b$)
case *True*
with *Normal exec-mark*
have $\Gamma\vdash\langle$*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 $\Gamma\vdash\langle$*c1*,*Normal s'*$\rangle$ =$n\Rightarrow$ t'
isFault t $\longrightarrow$ *isFault t'*
t' = *Fault f* $\longrightarrow$ t'=t
isFault t' $\longrightarrow$ *isFault t*
$\neg$ *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 $\Gamma\vdash\langle$*mark-guards f c2* ,*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*,*Normal s'*$\rangle$ =$n\Rightarrow$ t'
isFault t $\longrightarrow$ *isFault t'*
t' = *Fault f* $\longrightarrow$ t'=t
isFault t' $\longrightarrow$ *isFault t*
$\neg$ *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*: $\Gamma\vdash\langle$*mark-guards f* (*While b c*),$s\rangle$ =$n\Rightarrow$ 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'*)
{
fix $c'\ r\ w$
assume *exec-c'*: $\Gamma\vdash\langle c',r\rangle =n\Rightarrow w$
assume *c'*: $c'=While\ b\ (mark\text{-}guards\ f\ c)$
have $\exists\, w'.\ \Gamma\vdash\langle While\ b\ c,r\rangle =n\Rightarrow w' \wedge (isFault\ w \longrightarrow isFault\ w') \wedge$
$(w' = Fault\ f \longrightarrow w'=w) \wedge (isFault\ w' \longrightarrow isFault\ w) \wedge$
$(\neg\ isFault\ w' \longrightarrow w'=w)$
using *exec-c' c'*
proof (*induct*)
case (*WhileTrue r b' c'' n u w*)
have *eqs*: $While\ b'\ c'' = While\ b\ (mark\text{-}guards\ f\ c)$ **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
        Γ⊢⟨While b c,u⟩ =n⇒ w'
        isFault w  ⟶ isFault w'
        w' = Fault f ⟶ w'=w
        isFault w' ⟶ isFault 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)
      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: $\Gamma\vdash\langle$*mark-guards f c*,*Normal s'*$\rangle$ $=n\Rightarrow$ *r* **and**
    exec-mark-w: $\Gamma\vdash\langle$ *While b* (*mark-guards f c*),*r*$\rangle$ $=n\Rightarrow$ *t*
    by (*auto elim*: *execn-Normal-elim-cases*)
  from *While.hyps exec-mark-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-f: *r'* = *Fault f* $\longrightarrow$ *r'*=*r* **and**
    r'-Fault: *isFault r'* $\longrightarrow$ *isFault r* **and**
    r'-noFault: ¬ *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-mark-w*
    obtain *t'* **where**
      $\Gamma\vdash\langle$ *While b c*,*r*$\rangle$ $=n\Rightarrow$ *t'*
      isFault t $\longrightarrow$ *isFault t'*
      t' = *Fault f* $\longrightarrow$ *t'*=*t*
      isFault t' $\longrightarrow$ *isFault t*
      ¬ *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*)
    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*: $\Gamma\vdash\langle$*mark-guards f* (*Guard* f' *g c*),*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* $s'\in g$)
case *False*
with *Normal exec-mark* **have** *t*: *t*=*Fault f*
by (*auto elim*: *execn-Normal-elim-cases*)
from *False*
have $\Gamma\vdash\langle$*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 $\Gamma\vdash\langle$*mark-guards f c*,*Normal* $s'\rangle$ $=n\Rightarrow$ *t*
by (*auto elim*: *execn-Normal-elim-cases*)
with *Guard.hyps* **obtain** t' **where**
$\Gamma\vdash\langle c,$*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*: $\Gamma\vdash\langle$*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**
$\Gamma\vdash\langle$*Catch* (*mark-guards f c1*) (*mark-guards f c2*),*Normal s'*$\rangle$ =*n*$\Rightarrow$ *t* **by** *simp*
thus *?thesis*
proof (*cases*)
fix *w*
assume *exec-mark-c1*: $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s'*$\rangle$ =*n*$\Rightarrow$ *Abrupt w*
assume *exec-mark-c2*: $\Gamma\vdash\langle$*mark-guards f c2*,*Normal w*$\rangle$ =*n*$\Rightarrow$ *t*
from *exec-mark-c1 Catch.hyps*
obtain *w'* **where**
exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s'*$\rangle$ =*n*$\Rightarrow$ *w'* **and**
w'-Fault-f: *w'* = *Fault f* $\longrightarrow$ *w'*=*Abrupt w* **and**
w'-Fault: *isFault w'* $\longrightarrow$ *isFault* (*Abrupt w*) **and**
w'-noFault: $\neg$ *isFault w'* $\longrightarrow$ *w'*=*Abrupt w*
by *fastforce*
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-mark-c2 Catch.hyps*
obtain *t*$'$ **where**
$\Gamma\vdash\langle$*c2*,*Normal w*$\rangle$ =*n*$\Rightarrow$ *t*$'$
isFault t $\longrightarrow$ *isFault t*$'$
t$'$ = *Fault f* $\longrightarrow$ *t*$'$=*t*
isFault t$'$ $\longrightarrow$ *isFault 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 $\Gamma\vdash\langle$*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*: $\neg$ *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-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*: $\neg$ *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-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-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

3.7 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-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-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*: $\Gamma\vdash\langle$*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*
  have *notinF*: $f \notin F$ **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 $\Gamma\vdash\langle$*strip-guards F c1*,*Normal s*$\rangle$ $=n\Rightarrow$ *Fault f*
      by *auto*
    moreover have $\Gamma\vdash\langle$*strip-guards F 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-strip-guards* [*OF exec-c1*]
    have *exec-strip-c1*: $\Gamma\vdash\langle$*strip-guards F c1*,*Normal s*$\rangle$ $=n\Rightarrow$ 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*: Γ⊢⟨*c1*,*Normal s*⟩ =*n*⇒ *Abrupt w* **by** *fact*
have *exec-c2*: Γ⊢⟨*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⊢⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *Abrupt w*
by *simp*
with *CatchMatch.hyps t notinF*
have Γ⊢⟨*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*)
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-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-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'*) $\wedge$
          ($t' \in$ *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*: $\Gamma\vdash\langle$*strip-guards F* (*Seq c1 c2*),$s\rangle =n\Rightarrow t$ **by** *fact*
  then obtain *w* **where**
    exec-strip-c1: $\Gamma\vdash\langle$*strip-guards F c1*,$s\rangle =n\Rightarrow w$ **and**
    exec-strip-c2: $\Gamma\vdash\langle$*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: $\Gamma\vdash\langle c1,s\rangle =n\Rightarrow w'$ **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*: $\Gamma\vdash\langle$*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**
$\Gamma\vdash\langle$*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: 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*: $\Gamma\vdash\langle$*strip-guards F c*,*Normal r*$\rangle$ $=n\Rightarrow$ *u* **by** *simp*
from *WhileTrue.hyps eqs*
have *exec-strip-w*: $\Gamma\vdash\langle$*While b* (*strip-guards F c*),*u*$\rangle$ $=n\Rightarrow$ *w*
  by *simp*
show *?case*
proof −
  from *WhileTrue.hyps eqs* **have** $\Gamma\vdash\langle$*strip-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: *u'* $\in$ *Fault* ‘ $(-F)$ $\longrightarrow$ *u'*=*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' $\in$ *Fault* ‘ $(-F)$ $\longrightarrow$ *w'*=*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*)
    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'* $\in$ *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: Γ⊢⟨strip-guards F c,Normal s'⟩ =n⇒ r and
    exec-strip-w: Γ⊢⟨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⊢⟨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
      Γ⊢⟨While b c,r⟩ =n⇒ t'
      isFault t ⟶ isFault t'
      t' ∈ Fault ` (−F) ⟶ t'=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 $\Gamma\vdash\langle$*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*: $\Gamma\vdash\langle$*strip-guards F* (*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*: $\Gamma\vdash\langle$*strip-guards* F c,*Normal* $s'\rangle$ $=n\Rightarrow$ t
by *simp*
with *Guard.hyps* obtain t' where
$\Gamma\vdash\langle c$,*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 $\Gamma\vdash\langle$*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 $\Gamma\vdash\langle$*strip-guards* F c,*Normal* $s'\rangle$ $=n\Rightarrow$ t
by (*auto elim*: *execn-Normal-elim-cases*)
with *Guard.hyps* obtain t' where
$\Gamma\vdash\langle c$,*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 (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
     Γ⊢⟨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⊢⟨strip-guards F c1,Normal s'⟩ =n⇒ Abrupt w
      assume exec-strip-c2: Γ⊢⟨strip-guards F c2,Normal w⟩ =n⇒ t
      from exec-strip-c1 Catch.hyps
      obtain w' where
        exec-c1: Γ⊢⟨c1,Normal s'⟩ =n⇒ w' and
        w'-Fault: w' ∈ Fault ` (−F) ⟶ w'=Abrupt w and
        w'-noFault: ¬ isFault w' ⟶ w'=Abrupt w
        by blast
      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-strip-c2 Catch.hyps
    obtain t' where
      Γ⊢⟨c2,Normal w⟩ =n⇒ t'
      isFault t ⟶ isFault t'
      t' ∈ Fault ` (−F) ⟶ t'=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⊢⟨strip-guards F c1,Normal s'⟩ =n⇒ t
  with Catch.hyps
  obtain t' where
    exec-c1: Γ⊢⟨c1,Normal s'⟩ =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 ?thesis
  proof (cases isFault t')
    case True
    then obtain f' where t': t'=Fault f'..
    with exec-c1 have Γ⊢⟨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
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: $\Gamma\vdash\langle c1, Normal\ s\rangle =n\Rightarrow r'$ **and**
    r'*-Fault*: $r' \in Fault$ ‘ $(-F) \longrightarrow r' = Abrupt\ r$ **and**
    r'*-noFault*: $\neg$ *isFault* $r' \longrightarrow r' = Abrupt\ r$ **and**
    exec-c2: $\Gamma\vdash\langle c2, Normal\ r\rangle =n\Rightarrow t'$ **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 *?case*
  proof (*cases isFault* r')
    case *True*
    then obtain f' **where** r': $r'=Fault\ f'$..
    with *exec-c1* **have** $\Gamma\vdash\langle Catch\ c1\ c2, Normal\ s\rangle =n\Rightarrow Fault\ f'$
      by (*auto intro*: *execn.intros*)
    with 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*: $\Gamma\vdash\langle strip\text{-}guards\ F\ c, s\rangle \Rightarrow t$
  shows $\exists t'.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t' \wedge$
    $(isFault\ t \longrightarrow isFault\ t') \wedge$
    $(t' \in Fault$ ‘ $(-F) \longrightarrow t'=t) \wedge$
    $(\neg\ isFault\ t' \longrightarrow t'=t)$
proof −
  from *exec-strip* **obtain** n **where**
    execn-strip: $\Gamma\vdash\langle strip\text{-}guards\ F\ c, s\rangle =n\Rightarrow t$
    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 $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle \Rightarrow t$
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 $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow t$
 by (*rule execn-to-execn-strip-guards*)
 thus $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle \Rightarrow t$
 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 $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle \Rightarrow t$
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 $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow t$
 by (*rule execn-to-execn-strip-guards'*)
 thus $\Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle \Rightarrow t$
 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*: $\neg$ *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* $\Gamma\vdash\langle bdy,Normal\ s\rangle$ $=n\Rightarrow$ *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*)
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
qed (*auto intro*: *execn.intros*)

lemma *exec-to-exec-strip*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
assumes *t-not-Fault*: $\neg$ *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

3.8 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*$\rangle =n\Rightarrow t$; $\neg$ *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 $\Gamma\vdash\langle c,$*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*)
  have (*Basic f* $\cap_g$ *c2*) = *Some c* **by** *fact*
  then obtain *c2*: *c2=Basic f* **and** *c*: *c=Basic f*
    by (*simp add*: *inter-guards-Basic*)
  have $\Gamma\vdash\langle c,$*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*)
  have (*Spec r* $\cap_g$ *c2*) = *Some c* **by** *fact*
  then obtain *c2*: *c2=Spec r* **and** *c*: *c=Spec r*
    by (*simp add*: *inter-guards-Spec*)

have $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ t$ **by** *fact*
with *c* **have** $\Gamma\vdash\langle Spec\ r,Normal\ s\rangle =n\Rightarrow\ t$ **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 $(Seq\ a1\ a2 \cap_g c2) = Some\ c$ **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 $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ t$ **by** *fact*
  with *c* **obtain** *s′* **where**
    exec-d1: $\Gamma\vdash\langle d1,Normal\ s\rangle =n\Rightarrow\ s'$ **and**
    exec-d2: $\Gamma\vdash\langle d2,s'\rangle =n\Rightarrow\ t$
    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
      $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow\ Normal\ s''$ **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: $(t1 \cap_g t2) = Some\ t3$ **and**
    e3: $(e1 \cap_g e2) = Some\ e3$ **and**
    c: $c=Cond\ b\ t3\ e3$
    by (*auto simp add*: *inter-guards-Cond*)
  have $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
  with *c* **have** $\Gamma\vdash\langle Cond\ b\ t3\ e3,Normal\ s\rangle =n\Rightarrow t$
    by *simp*
  then show *?case*
  proof (*cases*)
    assume *s-in-b*: $s\in b$
    assume $\Gamma\vdash\langle t3,Normal\ s\rangle =n\Rightarrow t$
    with *Cond.hyps t3 noFault*
    obtain $\Gamma\vdash\langle t1,Normal\ s\rangle =n\Rightarrow t$ $\Gamma\vdash\langle t2,Normal\ s\rangle =n\Rightarrow t$
      by *auto*
    with *s-in-b c2* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    assume *s-notin-b*: $s\notin b$
    assume $\Gamma\vdash\langle e3,Normal\ s\rangle =n\Rightarrow t$
    with *Cond.hyps e3 noFault*
    obtain $\Gamma\vdash\langle e1,Normal\ s\rangle =n\Rightarrow t$ $\Gamma\vdash\langle e2,Normal\ s\rangle =n\Rightarrow t$
      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: 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 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⊢⟨bdy1,Normal s⟩ =n⇒ Normal s'''
                 Γ⊢⟨bdy2,Normal s⟩ =n⇒ Normal s'''
            by auto
          moreover
          from Normal WhileTrue
          obtain
            Γ⊢⟨While b bdy1,Normal s'''⟩ =n⇒ s''
            Γ⊢⟨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⊢⟨bdy1,Normal s⟩ =n⇒ Abrupt s'''
               Γ⊢⟨bdy2,Normal s⟩ =n⇒ Abrupt s'''
          by auto
        moreover
        from Abrupt WhileTrue
        obtain
          Γ⊢⟨While b bdy1,Abrupt s'''⟩ =n⇒ s''
          Γ⊢⟨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 Γ⊢⟨bdy1,Normal s⟩ =n⇒ Stuck
               Γ⊢⟨bdy2,Normal s⟩ =n⇒ Stuck
          by auto
        moreover
        from Stuck WhileTrue
        obtain
          Γ⊢⟨While b bdy1,Stuck⟩ =n⇒ s''
          Γ⊢⟨While b bdy2,Stuck⟩ =n⇒ 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: ¬ isFault t by fact
  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* (λ*s*. *the* ((*f1 s*) $\cap_g$ (*f2 s*)))
by (*auto simp add*: *inter-guards-DynCom*)
have Γ⊢⟨*c*,*Normal s*⟩ =*n*⇒ *t* **by** *fact*
with *c* **have** Γ⊢⟨*DynCom* (λ*s*. *the* ((*f1 s*) $\cap_g$ (*f2 s*))),*Normal s*⟩ =*n*⇒ *t* **by** *simp*
then show *?case*
proof (*cases*)
assume *exec-f*: Γ⊢⟨*the* (*f1 s* $\cap_g$ *f2 s*),*Normal s*⟩ =*n*⇒ *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*: ¬ *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 $\neg$ *isAbr t*
moreover
assume $\Gamma\vdash\langle d1,Normal\ s\rangle =n\Rightarrow t$
with *d1 Catch.hyps noFault*
obtain $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow t$ **and** $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow t$
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*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
assumes *noFault*: $\neg$ *isFault t*
shows $\Gamma\vdash\langle c1,s\rangle =n\Rightarrow t \wedge \Gamma\vdash\langle c2,s\rangle =n\Rightarrow t$
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*: $(c1 \cap_g c2) = Some\ c$
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: $\Gamma\vdash\langle d1,Normal\ s\rangle =n\Rightarrow s'$ **and**
exec-d2: $\Gamma\vdash\langle d2,s'\rangle =n\Rightarrow Fault\ f$
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 $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*auto dest*: *Seq.hyps*)
thus *?thesis*
proof (*cases rule*: *disjE* [*consumes 1*])
assume $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f$
hence $\Gamma\vdash\langle Seq\ a1\ a2,Normal\ s\rangle =n\Rightarrow Fault\ f$
by (*auto intro*: *execn.intros*)
thus *?thesis*
by *simp*
next
assume $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
hence $\Gamma\vdash\langle Seq\ b1\ b2,Normal\ s\rangle =n\Rightarrow Fault\ f$
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: $\Gamma\vdash\langle a1, Normal\ s\rangle = n\Rightarrow Normal\ s''$ **and**
exec-b1: $\Gamma\vdash\langle b1, Normal\ s\rangle = n\Rightarrow Normal\ s''$
by *simp*
moreover from *d2 exec-d2 Normal*
have $\Gamma\vdash\langle a2, Normal\ s''\rangle = n\Rightarrow Fault\ f \vee \Gamma\vdash\langle b2, Normal\ s''\rangle = n\Rightarrow Fault\ f$
by (*auto dest*: *Seq.hyps*)
ultimately show *?thesis*
using *c2* **by** (*auto intro*: *execn.intros*)
qed
next
case (*Cond b t1 e1*)
have ($Cond\ b\ t1\ e1 \cap_g c2$) = *Some c* **by** *fact*
then obtain *t2 e2 t e* **where**
c2: *c2=Cond b t2 e2* **and**
t: ($t1 \cap_g t2$) = *Some t* **and**
e: ($e1 \cap_g e2$) = *Some e* **and**
c: *c=Cond b t e*
by (*auto simp add*: *inter-guards-Cond*)
have $\Gamma\vdash\langle c, Normal\ s\rangle = n\Rightarrow Fault\ f$ **by** *fact*
with *c* **have** $\Gamma\vdash\langle Cond\ b\ t\ e, Normal\ s\rangle = n\Rightarrow Fault\ f$ **by** *simp*
thus *?case*
proof (*cases*)
assume $s \in b$
moreover assume $\Gamma\vdash\langle t, Normal\ s\rangle = n\Rightarrow Fault\ f$
with *t* **have** $\Gamma\vdash\langle t1, Normal\ s\rangle = n\Rightarrow Fault\ f \vee \Gamma\vdash\langle t2, Normal\ s\rangle = n\Rightarrow Fault\ f$
by (*auto dest*: *Cond.hyps*)
ultimately show *?thesis* **using** *c2 c* **by** (*fastforce intro*: *execn.intros*)
next
assume $s \notin b$
moreover assume $\Gamma\vdash\langle e, Normal\ s\rangle = n\Rightarrow Fault\ f$
with *e* **have** $\Gamma\vdash\langle e1, Normal\ s\rangle = n\Rightarrow Fault\ f \vee \Gamma\vdash\langle e2, Normal\ s\rangle = n\Rightarrow Fault\ f$
by (*auto dest*: *Cond.hyps*)
ultimately show *?thesis* **using** *c2 c* **by** (*fastforce intro*: *execn.intros*)
qed
next
case (*While b bdy1*)
have ($While\ b\ bdy1 \cap_g c2$) = *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 $s \in g1 \cap g2$
moreover
assume $\Gamma\vdash\langle bdy,Normal\ s\rangle =n\Rightarrow Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Fault\ f$
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 $(Catch\ a1\ a2 \cap_g c2) = Some\ c$ **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 $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *fact*
with *c* **have** $\Gamma\vdash\langle Catch\ d1\ d2,Normal\ s\rangle =n\Rightarrow Fault\ f$ **by** *simp*
thus *?case*
proof (*cases*)
fix s'
assume $\Gamma\vdash\langle d1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
from *inter-guards-execn-noFault* [*OF d1 this*] **obtain**
exec-a1: $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$ **and**
exec-b1: $\Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
by *simp*
moreover assume $\Gamma\vdash\langle d2,Normal\ s'\rangle =n\Rightarrow Fault\ f$
with *d2*
have $\Gamma\vdash\langle a2,Normal\ s'\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle b2,Normal\ s'\rangle =n\Rightarrow Fault\ f$
by (*auto dest*: *Catch.hyps*)
ultimately show *?thesis*
using *c2* **by** (*fastforce intro*: *execn.intros*)
next
assume $\Gamma\vdash\langle d1,Normal\ s\rangle =n\Rightarrow Fault\ f$
with *d1* **have** $\Gamma\vdash\langle a1,Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle b1,Normal\ s\rangle =n\Rightarrow Fault\ f$
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*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow Fault\ f$
  shows $\Gamma\vdash\langle c1,s\rangle =n\Rightarrow Fault\ f \vee \Gamma\vdash\langle c2,s\rangle =n\Rightarrow Fault\ f$
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*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f$
  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

3.9 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 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: $\Gamma|_P\vdash\langle c1,Normal\ s\rangle =n\Rightarrow t'$ **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*: ¬ *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*:
$\llbracket\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\};\ s \in b\rrbracket$
$\Longrightarrow \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.WhileTrue*)

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

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

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

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

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

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

3.10 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*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
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 *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

4 Hoare Logic for Partial Correctness

theory *HoarePartialDef* **imports** *Semantic* **begin**

type-synonym $('s,'p)$ *quadruple* = $('s\ assn \times 'p \times 's\ assn \times 's\ assn)$

4.1 Validity of Hoare Tuples: $\Gamma,\Theta\models_{/F}\ P\ c\ Q,A$

definition
valid :: $[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] => bool$
(-$\models_{'/-}$/ - - -,- [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::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] =>bool$
(-,-$\models_{'/-}$/ - - -,- [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 :: $[('s,'p,'f)\ body,nat,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] => bool$
(-$\models$-:$_{'/-}$/ - - -,- [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::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,nat,'f\ set,$
$'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] \Rightarrow bool$
(-,-$\models$-:$_{'/-}$/ - - -,- [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*)

4.2 Properties of Validity

lemma *valid-iff-nvalid*: $\Gamma\models_{/F}$ *P c Q*,*A* = (∀ *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*: (∀ *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*:
⟦⋀*s t*. ⟦Γ⊢⟨*c*,*Normal s* ⟩ =*n*⇒ *t*;*s* ∈ *P*; *t*∉ *Fault* ‘ *F*⟧ $\Longrightarrow$ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*⟧
$\Longrightarrow$ $\Gamma\models n{:}_{/F}$ *P c Q*,*A*
by (*auto simp add*: *nvalid-def*)

lemma *validI*:
⟦⋀*s t*. ⟦Γ⊢⟨*c*,*Normal s* ⟩ ⇒ *t*;*s* ∈ *P*; *t*∉*Fault* ‘ *F*⟧ $\Longrightarrow$ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*⟧
$\Longrightarrow$ $\Gamma\models_{/F}$ *P c Q*,*A*
by (*auto simp add*: *valid-def*)

lemma *cvalidI*:
⟦⋀*s t*. ⟦∀ (*P*,*p*,*Q*,*A*)∈Θ. $\Gamma\models_{/F}$ *P* (*Call p*) *Q*,*A*;Γ⊢⟨*c*,*Normal s*⟩ ⇒ *t*;*s* ∈ *P*;*t*∉*Fault* ‘ *F*⟧
$\Longrightarrow$ *t* ∈ *Normal* ‘ *Q* ∪ *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*;∀ (*P*,*p*,*Q*,*A*)∈Θ. $\Gamma\models_{/F}$ *P* (*Call p*) *Q*,*A*;Γ⊢⟨*c*,*Normal s*⟩ ⇒ *t*;*s* ∈ *P*;*t*∉*Fault* ‘ *F*⟧
$\Longrightarrow$ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*
by (*auto simp add*: *cvalid-def valid-def*)

lemma *cnvalidI*:
⟦⋀*s t*. ⟦∀ (*P*,*p*,*Q*,*A*)∈Θ. $\Gamma\models n{:}_{/F}$ *P* (*Call p*) *Q*,*A*;
Γ⊢⟨*c*,*Normal s* ⟩ =*n*⇒ *t*;*s* ∈ *P*;*t*∉*Fault* ‘ *F*⟧
$\Longrightarrow$ *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*⟧
$\Longrightarrow$ $\Gamma,\Theta\models n{:}_{/F}$ *P c Q*,*A*
by (*auto simp add*: *cnvalid-def nvalid-def*)

lemma *cnvalidD*:

$[\![\Gamma,\Theta \models n:_{/F}\ P\ c\ Q,A; \forall (P,p,Q,A) \in \Theta.\ \Gamma \models n:_{/F}\ P\ (Call\ p)\ Q,A;$
$\Gamma \vdash \langle c, Normal\ s\ \rangle =n\Rightarrow\ t; s \in P;$
$t \notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A$
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' $\Gamma \vdash \langle c, Normal\ s\ \rangle =n\Rightarrow\ t$
assume P: $s \in P$
assume F: $t \notin Fault\ `\ F$
from *exec-strip* **obtain** t' **where**

exec: $\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow\ t'$ **and**
t': $t' \in Fault$ ‘ $(-F') \longrightarrow t'{=}t$ ¬ *isFault* $t' \longrightarrow t'{=}t$
by (*blast dest*: *execn-strip-to-execn*)
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
proof (*cases* $t' \in Fault$ ‘ F)
case *True*
with t' F F' **have** *False*
by *blast*
thus *?thesis* **..**
next
case *False*
with *exec P validn*
have $*$: $t' \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
by (*auto simp add*: *nvalid-def*)
with t' **have** $t'{=}t$
by *auto*
with $*$ **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' $\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow\ t$
assume P: $s \in P$
assume F: $t \notin Fault$ ‘ F
from *exec-strip* **obtain** t' **where**
exec: $\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow\ t'$ **and**
t': $t' \in Fault$ ‘ $(-F') \longrightarrow t'{=}t$ ¬ *isFault* $t' \longrightarrow t'{=}t$
by (*blast dest*: *exec-strip-to-exec*)
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
proof (*cases* $t' \in Fault$ ‘ 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*)
with t' **have** $t'{=}t$
by *auto*
with $*$ **show** *?thesis*

by *simp*
qed
qed

4.3 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-,-/$\vdash_{'/-}$ (-/ (-)/ -,/-)) [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*: $\Gamma,\Theta\vdash_{/F}\ (g \cap P)\ c\ Q,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ (g \cap P)\ (\mathit{Guard}\ f\ g\ c)\ Q,A$

| *Guarantee*: $[\![f \in F;\ \Gamma,\Theta\vdash_{/F}\ (g \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$

| *CallRec*:
$[\![(P,p,Q,A) \in \mathit{Specs};$
$\forall (P,p,Q,A) \in \mathit{Specs}.\ p \in \mathit{dom}\ \Gamma \wedge \Gamma,\Theta\cup\mathit{Specs}\vdash_{/F}\ P\ (\mathit{the}\ (\Gamma\ p))\ Q,A\ ]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$

| *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* Θ *F* Θ')
  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* Θ *F c Q A* Θ')
  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* Θ *F P c Q A* Θ')
  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*)+

4.4 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'=λZ. P'* **and** *?Q'=λZ. 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

5 Properties of Partial Correctness Hoare Logic

theory *HoarePartialProps* **imports** *HoarePartialDef* **begin**

5.1 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 Γ⊢⟨*Skip*,*Normal s*⟩ =*n*⇒ *t s* ∈ *P*
    thus *t* ∈ *Normal* ‘ *P* ∪ *Abrupt* ‘ *A*
      by *cases auto*
  qed
next
  case (*Basic* Θ *F f P A*)
  show $\Gamma,\Theta \models n{:}_{/F}$ {*s*. *f s* ∈ *P*} (*Basic f*) *P*,*A*
  proof (*rule cnvalidI*)
    fix *s t*
    assume Γ⊢⟨*Basic f*,*Normal s*⟩ =*n*⇒ *t s* ∈ {*s*. *f s* ∈ *P*}
    thus *t* ∈ *Normal* ‘ *P* ∪ *Abrupt* ‘ *A*
      by *cases auto*
  qed
next
  case (*Spec* Θ *F r Q A*)
  show $\Gamma,\Theta\models n{:}_{/F}$ {*s*. (∀ *t*. (*s*, *t*) ∈ *r* ⟶ *t* ∈ *Q*) ∧ (∃ *t*. (*s*, *t*) ∈ *r*)} *Spec r Q*,*A*
  proof (*rule cnvalidI*)
    fix *s t*
    assume *exec*: Γ⊢⟨*Spec r*,*Normal s*⟩ =*n*⇒ *t*
    assume *P*: *s* ∈ {*s*. (∀ *t*. (*s*, *t*) ∈ *r* ⟶ *t* ∈ *Q*) ∧ (∃ *t*. (*s*, *t*) ∈ *r*)}
    from *exec P*
    show *t* ∈ *Normal* ‘ *Q* ∪ *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 ∈ Normal ‘ (P ∩ − b) ∪ 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 Γ⊢⟨While b′ c′,r⟩ =n⇒ 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: ⋀n. Γ,Θ ⊨n:/F (g ∩ P) c Q,A by fact
  show Γ,Θ ⊨n:/F (g ∩ P) Guard f g 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⊢⟨Guard f g c,Normal s⟩ =n⇒ t
    assume t-notin-F: t ∉ Fault ‘ F
    assume P:s ∈ (g ∩ P)
    from exec P have Γ⊢⟨c,Normal s⟩ =n⇒ t
      by cases auto
    from valid-c ctxt this P t-notin-F
    show t ∈ Normal ‘ Q ∪ 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*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models m:_{/F}\ P\ (Call\ p)\ Q,A$
            $\Longrightarrow \forall (P,p,Q,A) \in Specs.\ \Gamma \models m:_{/F}\ P\ (Call\ p)\ Q,A$ **by** *fact*
        have $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models Suc\ m:_{/F}\ P\ (Call\ p)\ Q,A$ **by** *fact*
        hence *ctxt-m*: $\forall (P,\ p,\ Q,\ A) \in \Theta.\ \Gamma \models m:_{/F}\ P\ (Call\ p)\ Q,A$
          by (*fastforce simp add*: *nvalid-def intro*: *execn-Suc*)
        hence *valid-Proc*:

$\forall (P,p,Q,A) \in Specs.\ \Gamma \models m{:}_{/F}\ P\ (Call\ p)\ Q,A$
by (*rule hyp*)
let $?\Theta' = \Theta \cup Specs$
from *valid-Proc ctxt-m*
have $\forall (P,\ p,\ Q,\ A) \in ?\Theta'.\ \Gamma \models m{:}_{/F}\ P\ (Call\ p)\ Q,A$
by *fastforce*
with *valid-body*
have *valid-body-m*:
$\forall (P,p,Q,A) \in Specs.\ \forall n.\ \Gamma \models m{:}_{/F}\ P\ (the\ (\Gamma\ p))\ Q,A$
by (*fastforce simp add*: *cnvalid-def*)
show $\forall (P,p,Q,A) \in Specs.\ \Gamma \models Suc\ m{:}_{/F}\ P\ (Call\ p)\ Q,A$
proof (*clarify*)
fix $P\ p\ Q\ A$ **assume** *p*: $(P,p,Q,A) \in Specs$
show $\Gamma \models Suc\ m{:}_{/F}\ P\ (Call\ p)\ Q,A$
proof (*rule nvalidI*)
fix $s\ t$
assume *exec-call*:
$\Gamma \vdash \langle Call\ p, Normal\ s\rangle = Suc\ m \Rightarrow t$
assume *Pre*: $s \in P$
assume *t-notin-F*: $t \notin Fault\ `\ F$
from *exec-call*
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
proof (*cases*)
fix $bdy\ m'$
assume *m*: $Suc\ m = Suc\ m'$
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume *exec-body*: $\Gamma \vdash \langle bdy, Normal\ s\rangle = m' \Rightarrow t$
from *Pre valid-body-m exec-body bdy m p t-notin-F*
show *?thesis*
by (*fastforce simp add*: *nvalid-def*)
next
assume $\Gamma\ p = None$
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 $(DynCom\ P\ \Theta\ F\ c\ Q\ A)$
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*: $t \notin Fault$ ‘ F
from *exec* **show** $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ 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 $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ 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*: $t \notin Fault$ ‘ F
from *exec* **show** $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
proof (*cases*)
fix s'
assume *exec-c1*: $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
assume *exec-c2*: $\Gamma\vdash\langle c_2,Normal\ s'\rangle =n\Rightarrow t$
from *cnvalidD* [*OF valid-c1 ctxt exec-c1 P*]
have $Abrupt\ s' \in Abrupt$ ‘ R
by *auto*
with *cnvalidD* [*OF valid-c2 ctxt - - t-notin-Fault*] *exec-c2*
show *?thesis*
by *fastforce*
next
assume *exec-c1*: $\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow t$
assume *notAbr*: $\neg\ isAbr\ t$
from *cnvalidD* [*OF valid-c1 ctxt exec-c1 P t-notin-Fault*]
have $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ R **.**
with *notAbr*

show *?thesis*
  by *auto*
qed
qed
next
case (*Conseq* P Θ F c Q A)
hence *adapt*: $\forall\, s \in P.\ (\exists\, P'\ Q'\ A'.\ \Gamma,\Theta \models n{:}_{/F}\ P'\ c\ Q',A'\ \wedge$
$s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
show $\Gamma,\Theta \models n{:}_{/F}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
fix s t
assume *ctxt*: $\forall\, (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models}n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
assume *exec*: $\Gamma{\vdash}\langle c,\mathit{Normal}\ s\rangle\ ={n}{\Rightarrow}\ t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin \mathit{Fault}\ `\ F$
show $t \in \mathit{Normal}\ `\ Q \cup \mathit{Abrupt}\ `\ A$
proof −
from *P adapt* **obtain** $P'\ Q'\ A'\ Z$ **where**
spec: $\Gamma,\Theta{\models}n{:}_{/F}\ P'\ c\ Q',A'$ **and**
P': $s \in P'$ **and** *strengthen*: $Q' \subseteq Q \wedge A' \subseteq A$
by *auto*
from *spec* [*rule-format*] *ctxt exec* P' *t-notin-F*
have $t \in \mathit{Normal}\ `\ Q' \cup \mathit{Abrupt}\ `\ A'$
by (*rule cnvalidD*)
with *strengthen* **show** *?thesis*
by *blast*
qed
qed
next
case (*Asm* P p Q A Θ F)
have *asm*: $(P,\ p,\ Q,\ A) \in \Theta$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
proof (*rule cnvalidI*)
fix s t
assume *ctxt*: $\forall\, (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
assume *exec*: $\Gamma{\vdash}\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle\ ={n}{\Rightarrow}\ t$
from *asm ctxt* **have** $\Gamma \models n{:}_{/F}\ P\ \mathit{Call}\ p\ Q,A$ **by** *auto*
moreover
assume $s \in P$ $t \notin \mathit{Fault}\ `\ F$
ultimately
show $t \in \mathit{Normal}\ `\ Q \cup \mathit{Abrupt}\ `\ A$
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*)

5.2 Completeness

lemma *MGT-valid*:
$\Gamma\models_{/F}\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup\ \ Fault\ `\ (-F))\}\ c$
  $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$, $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule validI*)
  fix *s t*
  assume $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
    $s \in \{s.\ s = Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup\ \ Fault\ `\ (-F))\}$
    $t \notin Fault\ `\ F$
  thus $t \in Normal\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\} \cup$
      $Abrupt\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
    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*: $\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ F)$
  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 $t \in$ *Fault* ‘ F
then obtain f **where**
t: t=*Fault* f **and** f: $f \in F$
by *auto*
from t *exec-c1*
have $\Gamma\vdash\langle$*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*: $\Gamma\vdash\langle$*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 *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 $\Gamma,\{\} \vdash_{/F} P\ c\ Q,A$
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 Γ,{} $\vdash_{/F}$ {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*:
$\forall Z$. Γ,{}$\vdash_{/F}$ {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 Γ,{} $\vdash_{/F} P\ c\ Q,A$
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*: Γ,{} $\vdash_{/F} P\ c\ Q,A$
shows Γ,{}$\vdash_{/F}$ {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 ∃ t. (Z,t) ∈ r)
    apply (auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros)
    done
next
  case (Seq c1 c2)
  have hyp-c1: ∀ 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 Seq.hyps by iprover
  have hyp-c2: ∀ 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 Seq.hyps by iprover
  from hyp-c1
  have Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨Seq c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
c1
            {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Normal t ∧
                Γ⊢⟨c2,Normal t⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))},
            {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Abrupt t}
    by (rule ConseqMGT)
      (auto dest: Seq-NoFaultStuckD1 [simplified] Seq-NoFaultStuckD2 [simplified]
          intro: exec.Seq)
  thus Γ,Θ⊢/F {s. s=Z ∧ Γ⊢⟨Seq c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
                Seq c1 c2
            {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Normal t},
            {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Abrupt t}
  proof (rule hoarep.Seq )
    show Γ,Θ⊢/F {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Normal t ∧
                    Γ⊢⟨c2,Normal t⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
                c2
              {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Normal t},
              {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Abrupt t}
    proof (rule ConseqMGT [OF hyp-c2],safe)
      fix r t
      assume Γ⊢⟨c1,Normal Z⟩ ⇒ Normal r Γ⊢⟨c2,Normal r⟩ ⇒ Normal t
      then show Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ 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
```

$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}$ $(\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}\cap b)$
$c1$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.CondTrue simp add*: *final-notin-def*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}(\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}\cap-b)$
$c2$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.CondFalse simp add*: *final-notin-def*)
ultimately
show $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$Cond\ b\ c1\ c2$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoarep.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 $?P' = \lambda Z.\ \{t.\ (Z,t)\in ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u))\}$
let $?A' = \lambda Z.\ \{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
show $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$While\ b\ c$
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [**where** $?P'=?P'$
and $?Q'=\lambda Z.\ ?P'\ Z \cap -\ b$ **and** $?A'=?A'$])
show $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $(?P'\ Z)$ $(While\ b\ c)$ $(?P'\ Z \cap -\ b),(?A'\ Z)$
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*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Abrupt\ t$
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*: $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
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*: $[\![e{\in}b;\Gamma\vdash\langle$*While b c*,*Normal r*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))]\!]$
$\Longrightarrow$ *?Prop r e*
assume *Z-r*:
$(Z,\ r) \in \{(Z,\ r).\ Z \in b \wedge \Gamma\vdash\langle c,$*Normal Z*$\rangle \Rightarrow$ *Normal r*$\}$
with *WhileNoFault*
have $\Gamma\vdash\langle$*While b c*,*Normal r*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
from *hyp* [*OF e-in-b this*] **obtain**
cNoFault: $\Gamma\vdash\langle c,$*Normal e*$\rangle \Rightarrow\notin(\{$*Stuck*$\} \cup$ *Fault* ‘ $(-F))$ **and**
Abrupt-r: $\forall\, u.\ \Gamma\vdash\langle c,$*Normal e*$\rangle \Rightarrow$ *Abrupt u* $\longrightarrow$
$\Gamma\vdash\langle$*While b c*,*Normal r*$\rangle \Rightarrow$ *Abrupt u*
by *simp*

{
fix u **assume** $\Gamma\vdash\langle c,$*Normal e*$\rangle \Rightarrow$ *Abrupt u*
with *Abrupt-r* **have** $\Gamma\vdash\langle$*While b c*,*Normal r*$\rangle \Rightarrow$ *Abrupt u* **by** *simp*
moreover from *Z-r* **obtain**
$Z \in b$ $\Gamma\vdash\langle c,$*Normal Z*$\rangle \Rightarrow$ *Normal r*
by *simp*
ultimately have $\Gamma\vdash\langle$*While b c*,*Normal Z*$\rangle \Rightarrow$ *Abrupt u*
by (*blast intro*: *exec.intros*)
}
with *cNoFault* **show** *?Prop Z e*
by *iprover*
qed
}
with P **show** $s \in$ *?P′ s*
by *blast*
next
{
fix t
assume *termination*: $t \notin b$
assume $(Z,\ t) \in$ *?unroll*
hence $\Gamma\vdash\langle$*While b c*,*Normal Z*$\rangle \Rightarrow$ *Normal t*
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
from *termination*
show $\Gamma\vdash\langle$*While b c*,*Normal t*$\rangle \Rightarrow$ *Normal t*
by (*blast intro*: *exec.WhileFalse*)
next
fix Z r
assume *first-body*:
$(Z,\ r) \in \{(s,\ t).\ s \in b \wedge \Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow$ *Normal t*$\}$
assume $(r,\ t) \in$ *?unroll*
assume *rest-loop*: $\Gamma\vdash\langle$*While b c*, *Normal r*$\rangle \Rightarrow$ *Normal t*
show $\Gamma\vdash\langle$*While b c*,*Normal Z*$\rangle \Rightarrow$ *Normal t*

proof −
from *first-body* **obtain**
$Z \in b\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ r$
by *fast*
moreover
from *rest-loop* **have**
$\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow Normal\ t$
by *fast*
ultimately show $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*rule exec.WhileTrue*)
qed
qed
}
with *P*
show $(\forall t.\ t\in(?P'\ s \cap -\ b)$
$\longrightarrow t\in\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\})$
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*)
let $?P = \{s.\ s=Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
from *noStuck-Call* **have** $\forall s \in ?P.\ p \in dom\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show $\Gamma,\Theta\vdash_{/F}\ ?P\ (Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule conseq-extract-state-indep-prop*)
assume *p-definied*: $p \in dom\ \Gamma$
with *MGT-Calls* **show**
$\Gamma,\Theta\vdash_{/F}\{s.\ s=Z \wedge$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
$(Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ \ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*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 $\Gamma,\Theta\vdash_{/F}$
$(g \cap \{s.\ s=Z \wedge \Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\})$
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 *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 $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))\}$
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 hoarep.Throw*]) (*blast intro*: *exec.intros*)
next
case (*Catch* c_1 c_2)
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))\}$
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*
hence $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}$
$(-F))\}\ c_1$
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t \wedge$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))\}$
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*
hence $\Gamma,\Theta\vdash_{/F}\{s.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ s \wedge$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))\}$
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 $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}$
$(-F))\}$
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*:
$\forall\, p{\in}dom\ \Gamma.\ \forall\, Z.$
$\quad\Gamma,\{\}\vdash_{/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ ({-}F))\}$
$\qquad(Call\ p)$
$\qquad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\qquad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof −
{
fix $p\ Z$
assume *defined*: $p \in dom\ \Gamma$
have
$\Gamma,(\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.$
$\quad\{(\{s.\ s{=}Z \wedge$
$\qquad\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ ({-}F))\},$
$\qquad p,$
$\qquad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\qquad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\})\})$
$\ \vdash_{/F}\{s.\ s = Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ ({-}F))\}$
$\quad(the\ (\Gamma\ p))$
$\quad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\quad\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
(**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 $\forall\, Z.\ \Gamma,?\Theta\vdash_{/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle the\ (\Gamma\ p)\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault`({-}F))\}$
$\quad(the\ (\Gamma\ p))$
$\quad\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\quad\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*iprover intro*: *MGT-lemma* [*OF MGT-Calls*])
thus $\Gamma,?\Theta\vdash_{/F} (?Pre\ p\ Z)\ (the\ (\Gamma\ p))\ (?Post\ p\ Z),(?Abr\ p\ Z)$
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** $\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ ({-}F))$
by (*fastforce simp add*: *final-notin-def*
intro: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Normal\ t$

```
        with defined
        show Γ⊢⟨Call p,Normal Z⟩ ⇒Normal t
          by (auto intro: exec.Call)
      next
        fix t
        assume Γ⊢⟨the (Γ p),Normal Z⟩ ⇒ Abrupt t
        with defined
        show Γ⊢⟨Call p,Normal Z⟩ ⇒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=λp Z. {s. s=Z ∧
               Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}and
      Q=λp Z.
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t} and
      A=λp Z.
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t}] )
    apply simp+
    done
qed

theorem hoare-complete: Γ⊨/F P c Q,A ⟹ Γ,{}⊢/F P c Q,A
  by (iprover intro: MGT-implies-complete MGT-lemma [OF MGT-Calls])

lemma hoare-complete′:
  assumes cvalid: ∀ n. Γ,Θ⊨n:/F P c Q,A
  shows  Γ,Θ⊢/F P c Q,A
proof (cases Γ⊨/F P c Q,A)
  case True
  hence Γ,{}⊢/F P c Q,A
    by (rule hoare-complete)
  thus Γ,Θ⊢/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,{}⊢/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

5.3 And Now: Some Useful Rules

5.3.1 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 $\Gamma,\Theta \models n:_{/F} P\ c\ Q,A$
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 −
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 ∪ *procs* c_2)

primrec *noSpec*:: ('*s*,'*p*,'*f*) *com* ⇒ *bool*
where
noSpec Skip = *True* |
noSpec (*Basic f*) = *True* |
noSpec (*Spec r*) = *False* |
noSpec (*Seq* c_1 c_2) = (*noSpec* c_1 ∧ *noSpec* c_2) |
noSpec (*Cond b* c_1 c_2) = (*noSpec* c_1 ∧ *noSpec* c_2) |
noSpec (*While b c*) = *noSpec c* |
noSpec (*Call p*) = *True* |
noSpec (*DynCom c*) = (∀ *s*. *noSpec* (*c s*)) |
noSpec (*Guard f g c*) = *noSpec c* |
noSpec Throw = *True* |
noSpec (*Catch* c_1 c_2) = (*noSpec* c_1 ∧ *noSpec* c_2)

lemma *exec-noSpec-no-Stuck*:
assumes *exec*: Γ⊢⟨*c*,*s*⟩ ⇒ *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 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*: $\forall Z. \forall n. \Gamma,\Theta \models n:_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
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*: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-Γ*: $\forall p \in dom\ \Gamma.\ nothrows\ (the\ (\Gamma\ p))$
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-Γ*: $\forall p \in dom\ \Gamma.\ noSpec\ (the\ (\Gamma\ p))$
assumes *procs-subset*: $procs\ c \subseteq dom\ \Gamma$
assumes *procs-subset-Γ*: $\forall p \in dom\ \Gamma.\ procs\ (the\ (\Gamma\ p)) \subseteq dom\ \Gamma$
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 −
    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 $(\forall Z. s \in P'\ Z \longrightarrow t' \in Q'\ Z)$
      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*: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-Γ*: $\forall p \in dom\ \Gamma.\ nothrows\ (the\ (\Gamma\ p))$
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-Γ*: $\forall p \in dom\ \Gamma.\ noSpec\ (the\ (\Gamma\ p))$
assumes *procs-subset*: $procs\ c \subseteq dom\ \Gamma$

assumes *procs-subset-*Γ: $\forall p \in dom\ \Gamma.\ procs\ (the\ (\Gamma\ p)) \subseteq dom\ \Gamma$
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*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\{s.\ s{=}fst\ Z \wedge P\ s\ (snd\ Z)\}\ c\ \{t.\ Q\ (fst\ Z)\ (snd\ Z)\ t\},\{\}$
assumes *noguards-c*: *noguards c*
assumes *noguards-*Γ: $\forall p \in dom\ \Gamma.\ noguards\ (the\ (\Gamma\ p))$
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: $\forall p \in dom\ \Gamma.\ nothrows\ (the\ (\Gamma\ p))$
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-*Γ: $\forall p \in dom\ \Gamma.\ noSpec\ (the\ (\Gamma\ p))$
assumes *procs-subset*: $procs\ c \subseteq dom\ \Gamma$
assumes *procs-subset-*Γ: $\forall p \in dom\ \Gamma.\ procs\ (the\ (\Gamma\ p)) \subseteq dom\ \Gamma$
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

5.3.2 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'*: ∀ (P, p, Q, A)∈Θ. Γ $\models$n:$_{/UNIV}$ P (*Call* p) Q,A
  by (*auto intro*: *nvalid-augment-Faults*)
assume *exec*: Γ⊢⟨*call init p return c*,*Normal s*⟩ =n⇒ t
assume *P*: s ∈ P
assume *t-notin-F*: t ∉ *Fault* ‘ F
from *exec*
show t ∈ *Normal* ‘ Q ∪ *Abrupt* ‘ A
proof (*cases rule*: *execn-call-Normal-elim*)
  fix *bdy m t'*
  assume *bdy*: Γ p = *Some bdy*
  assume *exec-body*: Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =m⇒ *Normal t'*
  assume *exec-c*: Γ⊢⟨*c s t'*,*Normal* (*return s t'*)⟩ =*Suc m*⇒ t
  assume *n*: n = *Suc m*
  from *exec-body n bdy*
  have Γ⊢⟨*Call p*,*Normal* (*init s*)⟩ =n⇒ *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' ∈ *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' ∈ *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 $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Fault\ f\ n = Suc\ m$
    $t = Fault\ f$
  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*: Γ *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 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 *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*:
  $\forall\sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  shows $\Gamma,\Theta\vdash_{/F} P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule 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*:
    $\forall \sigma.\ \forall n.\ \Gamma,\Theta{\models}n{:}_{/F}\ \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  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*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  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.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 $\Gamma{\vdash}\langle call\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow t$
      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 $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Fault\ f\ n = Suc\ m$ **and**
t: $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
by (*auto intro*: *execn-callFault*)
from *cnvalidD* [*OF valid-call* [*rule-format*] *ctxt this P*] *t t-notin-F*
show *?thesis*
by *simp*
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 *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* 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 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

5.3.3 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* 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
  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*: $\Gamma\vdash\langle$*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′*: $\forall \sigma.\ \forall n.$
    $\Gamma,\Theta\models n{:}_{/UNIV} \{\sigma\}$ *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.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 $\Gamma\vdash\langle$*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*: Γ (*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*: 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* $(p\ s)$ $(Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*dynCall init p return c*) *Q*,*A*
apply (*rule hoare-complete'*)
apply (*rule allI*)

apply (*rule 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*:
  $\forall s \in P.\ \forall \sigma.\ \forall n.$
    $\Gamma,\Theta\models n{:}_{/F}\ \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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 $\Gamma,\Theta \models n{:}_{/F}\ P\ (dynCall\ init\ p\ return\ c)\ Q,A$
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'*: $\forall \sigma.\ \forall n.$
 $\Gamma,\Theta\models n{:}_{/F}\ \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  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*: $\Gamma\ (p\ s) = Some\ bdy$
  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*: Γ (*p s*) = *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 s*) ,*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′* ∈ *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 s*) *return′ c*,*Normal s*⟩ =*n*⇒ *t*
  by (*auto intro*: *execn-callAbrupt*)
hence Γ⊢⟨*dynCall init p return′ c*,*Normal s*⟩ =*n*⇒ *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 Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Fault f n* = *Suc m* **and**
  t: *t* = *Fault f*
with *bdy* **have** Γ⊢⟨*call init* (*p s*) *return′ c* ,*Normal s*⟩ =*n*⇒ *t*
  by (*auto intro*: *execn-callFault*)
hence Γ⊢⟨*dynCall init p return′ c*,*Normal s*⟩ =*n*⇒ *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*: Γ (*p s*) = *Some bdy*
assume Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Stuck n* = *Suc m*
  t = *Stuck*
with *bdy* **have** Γ⊢⟨*call init* (*p s*) *return′ c* ,*Normal s*⟩ =*n*⇒ *t*
  by (*auto intro*: *execn-callStuck*)
hence Γ⊢⟨*dynCall init p return′ c*,*Normal s*⟩ =*n*⇒ *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 *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* 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 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

5.3.4 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**.**
    then 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 **.**
    with ∗ **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

5.3.5 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

5.3.6 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 $\Gamma{\vdash}\langle c_1{,}Normal\ s\rangle\ ={n}{\Rightarrow}\ Fault\ f \lor \Gamma{\vdash}\langle c_2{,}Normal\ s\rangle\ ={n}{\Rightarrow}\ Fault\ f$
      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: (c₁ ∩g c₂) = Some c
  assumes deriv-c1: Γ,Θ⊢/F P c₁ Q,A
  assumes deriv-c2: Γ,Θ⊢/F P c₂ UNIV,UNIV
  shows Γ,Θ⊢/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: ∀ n. Γ,Θ⊨n:/F P c Q,A
  assumes valid-strip: ∀ n. Γ,Θ⊨n:/{} P (strip-guards (−F) c) UNIV,UNIV
  shows Γ,Θ⊨n:/{} P c Q,A
proof (rule cnvalidI)
  fix s t
  assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ⊨n:/{} P (Call p) Q,A
  hence ctxt': ∀ (P, p, Q, A)∈Θ. Γ⊨n:/F P (Call p) Q,A
    by (auto intro: nvalid-augment-Faults)
  assume exec: Γ⊢⟨c,Normal s⟩ =n⇒ t
  assume P: s ∈ 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 $f \notin -F$ **by** *simp*
with *exec Fault*
have $\Gamma\vdash\langle$*strip-guards* $(-F)$ *c*,*Normal s*$\rangle$ $=n\Rightarrow$ *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*: $\Gamma,\Theta\vdash_{/F}$ *P c Q*,*A*
assumes *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}$ *P* (*strip-guards* $(-F)$ *c*) *UNIV*,*UNIV*
shows $\Gamma,\Theta\vdash_{/\{\}}$ *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 ∈ F)
      case True
      hence f ∉ −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: Γ,Θ⊢/F P c Q,A
  assumes derivFlip: Γ,Θ⊢/−F P c UNIV,UNIV
  shows Γ,Θ⊢/{} 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: ∀ n. Γ,Θ⊨n:/{} P c Q,A
  shows Γ,Θ⊨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*: $\Gamma\vdash\langle mark\text{-}guards\ f\ c,Normal\ s\rangle =n\Rightarrow t$
  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 $\Gamma,\Theta\vdash_{/\{\}}\ P\ mark\text{-}guards\ f\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MarkGuardsI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MarkGuardsD-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n:_{/\{\}}\ P\ mark\text{-}guards\ f\ c\ Q,A$
  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-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-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*: $\Gamma{\vdash}\langle merge\text{-}guards\ c{,}Normal\ s\rangle\ {=}n{\Rightarrow}\ t$**.**
  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*: $\Gamma{,}\Theta{\vdash}_{/F}\ P\ merge\text{-}guards\ c\ Q{,}A$
  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

5.3.7 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

6 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'*:
$\llbracket P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q) \wedge (\forall t.\ t \in A'\ Z \longrightarrow t \in\ A)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z);$
$\forall Z.\ c\ Z = c\ \mathit{undefined}$
$\rrbracket \Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (\mathit{specAnno}\ P'\ c\ Q'\ A')\ Q,A$
apply (*simp only*: *subset-iff* [*THEN sym*])
apply (*erule* (*1*) *SpecAnno*)
apply *assumption*
done

lemma *SpecAnnoNoAbrupt*:
$\llbracket P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),\{\};$
$\forall Z.\ c\ Z = c\ \mathit{undefined}$
$\rrbracket \Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (\mathit{specAnno}\ P'\ c\ Q'\ (\lambda s.\ \{\}))\ Q,A$
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*:
$\llbracket P \subseteq \{s.\ (b\ s \longrightarrow f\ s{\in}Q) \wedge (\neg\ b\ s \longrightarrow g\ s{\in}Q)\}\rrbracket \Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ \mathit{Basic}\ (\lambda s.\ \mathit{if}\ b\ s\ \mathit{then}\ f\ s\ \mathit{else}\ g\ s)\ Q,A$
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*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ Spec\ (if\text{-}rel\ b\ f\ g\ h)\ Q,A$
apply (*rule Spec*)
apply (*auto simp add*: *if-rel-def*)
done

lemma *Seq* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Seq\ c_1\ c_2)\ Q,A$
by (*rule hoarep.Seq*)

lemma *SeqSwap*:
$[\![\Gamma,\Theta\vdash_{/F}\ R\ c2\ Q,A;\ \Gamma,\Theta\vdash_{/F}\ P\ c1\ R,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (Seq\ c1\ c2)\ Q,A$
by (*rule Seq*)

lemma *BSeq*:
$[\![\Gamma,\Theta\vdash_{/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (bseq\ c_1\ c_2)\ Q,A$
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* (λs 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* Γ
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* Γ
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* Γ
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*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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*
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*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ Call\ q\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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*
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*: $\Gamma,\Theta\vdash_{/F} P\ merge\text{-}guards\ c\ Q,A = \Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
  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'': $c'' = mark\text{-}guards\ False\ (strip\text{-}guards\ (-F)\ c')$
  assumes *c*: $merge\text{-}guards\ c = merge\text{-}guards\ (mark\text{-}guards\ False\ c')$
  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 $\Gamma,\Theta\vdash_{/\{\}} P\ (strip\text{-}guards\ (-\ F)\ c')\ UNIV,UNIV$
  by (*rule MarkGuardsD*)
  with *deriv*
  have $\Gamma,\Theta\vdash_{/\{\}} P\ c'\ Q,A$
  by (*rule CombineStrip*)
  hence $\Gamma,\Theta\vdash_{/\{\}} P\ mark\text{-}guards\ False\ c'\ Q,A$
  by (*rule MarkGuardsI*)
  hence $\Gamma,\Theta\vdash_{/\{\}} P\ merge\text{-}guards\ (mark\text{-}guards\ False\ c')\ Q,A$
  by (*rule MergeGuardsI*)
  hence $\Gamma,\Theta\vdash_{/\{\}} P\ merge\text{-}guards\ c\ Q,A$
  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

6.1 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*:
$[\![\Gamma,\Theta\vdash_{/F} P\ (\mathit{whileAnno}\ b\ I\ V\ c)\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ (\mathit{whileAnno}\ b\ J\ V\ c)\ Q,A$
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?*]: $\Gamma,\Theta\vdash_{/F} P\ \mathit{Skip}\ P,A$
by (*rule Skip*) *auto*

lemma *CondInt* [*trans*,*intro?*]:
$[\![\Gamma,\Theta\vdash_{/F} (P \cap b)\ c1\ Q,A;\ \Gamma,\Theta\vdash_{/F} (P \cap -\ b)\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (\mathit{Cond}\ b\ c1\ c2)\ Q,A$
by (*rule Cond*) *auto*

lemma *CondConj* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{/F} \{s.\ P\ s \wedge b\ s\}\ c1\ Q,A;\ \Gamma,\Theta\vdash_{/F} \{s.\ P\ s \wedge \neg\ b\ s\}\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} \{s.\ P\ s\}\ (\mathit{Cond}\ \{s.\ b\ s\}\ c1\ c2)\ Q,A$
by (*rule Cond*) *auto*

lemma *WhileInvInt* [*intro?*]:
$\Gamma,\Theta\vdash_{/F} (P \cap b)\ c\ P,A \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ (\mathit{whileAnno}\ b\ P\ V\ c)\ (P \cap -b),A$
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

7 Terminating Programs

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

7.1 Inductive Characterisation: $\Gamma\vdash c\downarrow s$

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

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

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

| *Guard*: $[\![s\in g;\ \Gamma\vdash c\downarrow(Normal\ s)]\!]$
  $\Longrightarrow$
  $\Gamma\vdash$*Guard f g c*$\downarrow$(*Normal s*)

| *GuardFault*: $s\notin g$
  $\Longrightarrow$
  $\Gamma\vdash$*Guard f g c*$\downarrow$(*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** $\Gamma\vdash\langle c2,Normal\ s''\rangle \Rightarrow t$
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*:
$[\![\Gamma\ p = \mathit{Some\ bdy};\ \Gamma\vdash \mathit{bdy} \downarrow \mathit{Normal}\ (\mathit{init}\ s);$
$\forall t.\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow \mathit{Normal}\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow \mathit{Normal}\ (\mathit{return}\ s\ t)]\!]$
$\Longrightarrow \Gamma\vdash \mathit{call\ init\ p\ return\ c} \downarrow \mathit{Normal}\ s$
apply (*unfold call-def*)
apply (*rule terminates-block*)
apply (*iprover intro*: *terminates.intros*)
apply (*auto elim*: *exec-Normal-elim-cases*)
done

lemma *terminates-callUndefined*:
$[\![\Gamma\ p = \mathit{None}]\!]$
$\Longrightarrow \Gamma\vdash \mathit{call\ init\ p\ return\ result} \downarrow \mathit{Normal}\ s$
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*: $\Gamma\vdash \mathit{call\ init\ p\ return\ c} \downarrow \mathit{Normal}\ s$
assumes *bdy*: $\bigwedge \mathit{bdy}.\ [\![\Gamma\ p = \mathit{Some\ bdy};\ \Gamma\vdash \mathit{bdy} \downarrow \mathit{Normal}\ (\mathit{init}\ s);$
$\forall t.\ \Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle \Rightarrow \mathit{Normal}\ t \longrightarrow \Gamma\vdash c\ s\ t \downarrow \mathit{Normal}\ (\mathit{return}\ s\ t)]\!]$
$\Longrightarrow P$
assumes *undef*: $[\![\Gamma\ p = \mathit{None}]\!] \Longrightarrow P$
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

7.2 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 $\forall s'.\ \Gamma\vdash\langle$*sequence Seq xs*,*Normal* $s''\rangle \Rightarrow s' \longrightarrow \Gamma\vdash$*sequence Seq ys* $\downarrow$
s'
by (*auto intro*: *exec.intros*)
from *Cons.hyps* [*OF termi-xs* [*simplified Normal*] *this*]
have $\Gamma\vdash$*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*:
$\bigwedge s.\ \Gamma\vdash$*sequence Seq* (*xs* @ *ys*) $\downarrow$ *Normal s*
$\Longrightarrow \Gamma\vdash$*sequence Seq xs* $\downarrow$ *Normal s* $\wedge$
($\forall s'.\ \Gamma\vdash\langle$*sequence Seq xs*,*Normal s* $\rangle \Rightarrow s' \longrightarrow\ \Gamma\vdash$*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*: $\Gamma\vdash$*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: 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)

7.3 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 $\Gamma\vdash$*Seq* (*strip-guards F c1*) (*strip-guards F c2*) $\downarrow$ *s* **by** *simp*
  thus $\Gamma\vdash$*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=*Abrupt* s' **thus** *?thesis* **by** *simp*
  next
    fix s'
    assume *s*: s=*Normal* s'
    assume $\Gamma\vdash$*strip-guards F c1* $\downarrow$ *Normal* s'
    hence $\Gamma\vdash$*c1* $\downarrow$ *Normal* s'
      by (*rule Seq.hyps*)
    moreover
    assume *c2*:
      $\forall s''.\ \Gamma\vdash\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow s'' \longrightarrow \Gamma\vdash$*strip-guards F c2*$\downarrow s''$
    {
      fix s'' **assume** *exec-c1*: $\Gamma\vdash\langle$*c1*,*Normal* $s'\rangle \Rightarrow s''$
      have $\Gamma\vdash$*c2* $\downarrow s''$
      proof (*cases* s'')
        case (*Normal* s''')
        with *exec-c1*
        have $\Gamma\vdash\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow s''$
          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 $\Gamma\vdash\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow s''$
          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 $\Gamma\vdash$*strip-guards F c1* $\downarrow$ *Normal* s'
  hence $\Gamma\vdash$*c1* $\downarrow$ *Normal* s'
    by (*rule Catch.hyps*)
  moreover
  assume *c2*:
    $\forall s''$. $\Gamma\vdash\langle$*strip-guards F c1*,*Normal* $s'\rangle \Rightarrow$ *Abrupt* s''
        $\longrightarrow \Gamma\vdash$*strip-guards F c2*$\downarrow$*Normal* s''
  {
    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-c1*
      have $\Gamma\vdash\langle$*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
qed

lemma *terminates-strip-to-terminates*:
  assumes *termi-strip*: *strip F* $\Gamma\vdash c\downarrow s$
  shows $\Gamma\vdash c\downarrow s$
using *termi-strip*
proof *induct*
  case (*Seq c1 s c2*)
  have $\Gamma\vdash$*c1* $\downarrow$ *Normal s* **by** *fact*
  moreover
  {
    fix s'
    assume *exec*: $\Gamma\vdash\langle$*c1*,*Normal s*$\rangle \Rightarrow s'$
    have $\Gamma\vdash$*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 Γ⊢c ↓ Normal s by fact
  moreover
  {
    fix s'
    assume exec: Γ⊢ ⟨c,Normal s⟩ ⇒ s'
    have Γ⊢While b c ↓ 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 Γ⊢c1 ↓ Normal s by fact
  moreover
  {
    fix s'
    assume exec: Γ⊢ ⟨c1,Normal s⟩ ⇒ Abrupt s'
    from exec-to-exec-strip [OF exec] Catch.hyps
    have Γ⊢c2 ↓ 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)
```

7.4 Lemmas about $c_1 \cap_g c_2$

```
lemma inter-guards-terminates:
  ⋀c c2 s. ⟦(c1 ∩g c2) = Some c; Γ⊢c1↓s ⟧
       ⟹ Γ⊢c↓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 $\Gamma\vdash\langle bdy1, Normal\ s\ \rangle \Rightarrow Normal\ t'$
by *simp*
with *WhileTrue* **have** $\Gamma\vdash While\ b\ bdy \downarrow Normal\ t'$
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 $(DynCom\ f1 \cap_g c2) = Some\ c$ **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: $c = DynCom\ (\lambda s.\ the\ ((f1\ s) \cap_g (f2\ s)))$
by (*auto simp add*: *inter-guards-DynCom*)
have *termi*: $\Gamma\vdash DynCom\ f1 \downarrow s$ **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*: $((f1\ s') \cap_g (f2\ s')) = Some\ f$
by *auto*
from *Normal termi*
have $\Gamma\vdash f1\ s' \downarrow (Normal\ s')$
by (*auto elim*: *terminates-Normal-elim-cases*)
from *DynCom.hyps f this*
have $\Gamma\vdash f \downarrow (Normal\ s')$
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 ∩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
```

$d1$: $(a1 \cap_g b1) = Some\ d1$ **and** $d2$: $(a2 \cap_g b2) = Some\ d2$ **and**
c: $c{=}Catch\ d1\ d2$
by (*auto simp add*: *inter-guards-Catch*)
have *termi-c1*: $\Gamma\vdash Catch\ a1\ a2 \downarrow s$ **by** *fact*
have $\Gamma\vdash Catch\ d1\ d2 \downarrow s$
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 $\Gamma\vdash d1 \downarrow Normal\ s'$
by (*auto elim*: *terminates-Normal-elim-cases intro*: *Catch.hyps*)
moreover
{
fix *t*
assume *exec-d1*: $\Gamma\vdash\langle d1,Normal\ s'\rangle \Rightarrow Abrupt\ t$
have $\Gamma\vdash d2 \downarrow Normal\ t$
proof −
from *inter-guards-exec-noFault* [*OF d1 exec-d1*]
have $\Gamma\vdash\langle a1,Normal\ s'\rangle \Rightarrow Abrupt\ t$
by *simp*
with *termi-c1 Normal-s* **have** $\Gamma\vdash a2 \downarrow Normal\ t$
by (*auto elim*: *terminates-Normal-elim-cases*)
with *d2* **have** $\Gamma\vdash d2 \downarrow Normal\ t$
by (*auto intro*: *Catch.hyps*)
with *Normal* **show** *?thesis* **by** *simp*
qed
}
ultimately have $\Gamma\vdash Catch\ d1\ d2 \downarrow Normal\ s'$
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: $(c1 \cap_g c2) = Some\ c$
assumes *termi-c2*: $\Gamma\vdash c2\downarrow s$
shows $\Gamma\vdash c\downarrow s$
proof −
from c **have** $(c2 \cap_g c1) = Some\ c$
by (*rule inter-guards-sym*)
from *this termi-c2* **show** *?thesis*
by (*rule inter-guards-terminates*)

qed

7.5 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: $\neg$ *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⊢mark-guards f c ↓ Normal s by fact
  moreover
  {
    fix t
    assume exec-mark: Γ⊢⟨mark-guards f c,Normal s ⟩ ⇒ t
    have Γ⊢mark-guards f (While b c) ↓ t
    proof −
      from exec-mark-guards-to-exec [OF exec-mark] obtain t' where
        exec-c1: Γ⊢⟨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 $\Gamma\vdash$*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 Γ⊢mark-guards f (Seq 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 ⟩ ⇒ 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*)

7.6 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 ⋀s. Γ⊢merge-guards c↓Normal s ⟹ Γ⊢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 Γ⊢merge-guards (Seq c1 c2) ↓ Normal s by fact
  then obtain
    termi-merge-c1: Γ⊢merge-guards c1 ↓ Normal s and
    termi-merge-c2: ∀s'. Γ⊢⟨merge-guards c1,Normal s ⟩ ⇒ s' ⟶
                     Γ⊢merge-guards 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 −
      from exec-to-exec-merge-guards [OF exec-c1]
      have Γ⊢⟨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⊢c' ↓ Normal u
    assume c': c' = merge-guards (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*: $s \in b$ **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** $\Gamma\vdash c \downarrow$ *Normal s*
by *auto*
moreover
have *hyp-w*: $\forall w.\ \Gamma\vdash\langle$*merge-guards c*,*Normal s* $\rangle \Rightarrow w \longrightarrow \Gamma\vdash$ *While b c* $\downarrow w$
using *WhileTrue* **by** *simp*
hence $\forall w.\ \Gamma\vdash\langle c$,*Normal s* $\rangle \Rightarrow w \longrightarrow \Gamma\vdash$ *While b c* $\downarrow w$
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*: $\Gamma\vdash$*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 $\Gamma\vdash$*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)

7.7 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*: $\Gamma\vdash Seq\ c1'\ c2' \downarrow Normal\ s$ **by** *fact*
  then obtain
    termi-c1': $\Gamma\vdash c1'\downarrow Normal\ s$ **and**
    termi-c2': $\forall s'.\ \Gamma\vdash\langle c1',Normal\ s\ \rangle \Rightarrow s' \longrightarrow \Gamma\vdash c2'\downarrow s'$
    by (*auto elim*: *terminates-Normal-elim-cases*)
  have *noFault*: $\Gamma\vdash\langle Seq\ c1'\ c2',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **by** *fact*
  hence *noFault-c1'*: $\Gamma\vdash\langle c1',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$
    by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
  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*
  from *termi-c1' c1-c1' noFault-c1'*
  have $\Gamma\vdash c1\downarrow Normal\ s$
    by (*rule Seq.hyps*)
  moreover
  {
    fix *t*
    assume *exec-c1*: $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow t$
    have $\Gamma\vdash c2\downarrow t$
    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-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': 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 Γ⊢⟨c2',Normal u ⟩ ⇒∉Fault ‘ UNIV
            by (auto intro: exec.intros simp add: final-notin-def)
          from termi-c2' [simplified Normal] c2-c2' this
          have Γ⊢c2 ↓ 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: Γ⊢⟨Cond b c1' c2',Normal s ⟩ ⇒∉Fault ‘ UNIV by fact
  have termi: Γ⊢Cond b c1' c2' ↓ Normal s 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
  thus ?case
  proof (cases s ∈ b)
    case True
    with termi have termi-c1': Γ⊢c1' ↓ Normal s
      by (auto elim: terminates-Normal-elim-cases)
    from True noFault have Γ⊢⟨c1',Normal s ⟩ ⇒∉Fault ‘ UNIV
      by (auto intro: exec.intros simp add: final-notin-def)
    from termi-c1' c1-c1' this
    have Γ⊢c1 ↓ 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': Γ⊢⟨c',Normal u ⟩ ⇒ v' and
          v-Fault: isFault v ⟶ isFault v' and
          v'-noFault: ¬ isFault v' ⟶ 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 $\Gamma\vdash C\ s \downarrow Normal\ s$
  by *fast*
with *c* **show** *?case*
  by (*auto intro*: *terminates.intros*)
next
case (*Guard f′ g′ c′*)
have *noFault*: $\Gamma\vdash\langle Guard\ f'\ g'\ c',Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ UNIV$ **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 $[\![\Gamma\vdash c'\downarrow s;\ c \subseteq_g c';\ \Gamma\vdash\langle c',s\ \rangle \Rightarrow\notin Fault\ `\ UNIV]\!]$
    $\Longrightarrow \Gamma\vdash c\downarrow s$
  by (*cases s*) (*auto intro*: *terminates-fewer-guards-Normal*)

lemma *terminates-noFault-strip-guards*:
  assumes *termi*: $\Gamma\vdash c\downarrow Normal\ s$
  shows $[\![\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F]\!] \Longrightarrow \Gamma\vdash strip\text{-}guards\ F\ c\downarrow Normal\ s$
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** $\Gamma\vdash strip\text{-}guards\ F\ c \downarrow Normal\ s$ **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** $\Gamma\vdash strip\text{-}guards\ F\ c1 \downarrow Normal\ s$ **by** *simp*
  moreover
  {
    fix s'
    assume *exec-strip-guards-c1*: $\Gamma\vdash\langle strip\text{-}guards\ F\ c1,Normal\ s\ \rangle \Rightarrow s'$
    have $\Gamma\vdash strip\text{-}guards\ F\ c2 \downarrow s'$
    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 *: Γ⊢⟨c1,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      with noFault-Seq have Γ⊢⟨c2,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      with * 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⊢⟨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)
  with WhileTrue.hyps have Γ⊢strip-guards F c ↓ Normal s by simp
  moreover
  {
    fix s'
    assume exec-strip-guards-c: Γ⊢⟨strip-guards F c,Normal s ⟩ ⇒ s'
    have Γ⊢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⊢⟨c,Normal s ⟩ ⇒ s'
        by (auto simp add: final-notin-def elim!: isFaultE)
      with s-in-b noFault-while have Γ⊢⟨While b c,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      with * 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: Γ⊢⟨Catch c1 c2,Normal s ⟩ ⇒∉Fault ‘ F by fact
  hence noFault-c1: Γ⊢⟨c1,Normal s ⟩ ⇒∉Fault ‘ F
    by (fastforce simp add: final-notin-def intro: exec.intros)
  with Catch.hyps have Γ⊢strip-guards F c1 ↓ Normal s by simp
  moreover
  {
    fix s′
    assume exec-strip-guards-c1: Γ⊢⟨strip-guards F c1,Normal s ⟩ ⇒ Abrupt s′
    have Γ⊢strip-guards F c2 ↓ Normal s′
    proof −
      from exec-strip-guards-to-exec [OF exec-strip-guards-c1] noFault-c1
      have ∗: Γ⊢⟨c1,Normal s ⟩ ⇒ Abrupt s′
        by (auto simp add: final-notin-def elim!: isFaultE)
      with noFault-Catch have Γ⊢⟨c2,Normal s′ ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      with ∗ show ?thesis
        using Catch.hyps by simp
    qed
  }
  ultimately show ?case
    using Catch.hyps by (auto intro: terminates.intros)
qed
```

7.8 Lemmas about *strip-guards*

```
lemma terminates-noFault-strip:
  assumes termi: Γ⊢c↓Normal s
  shows ⟦Γ⊢⟨c,Normal s ⟩ ⇒∉Fault ‘ F⟧ =⇒ strip F Γ⊢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⊢⟨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)
      with noFault-Seq have Γ⊢⟨c2,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      with * 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)
      with s-in-b noFault-while have Γ⊢⟨While b c,s' ⟩ ⇒∉Fault ‘ F
        by (auto simp add: final-notin-def intro: exec.intros)
      with * 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*)
      with ∗ *noFault-Catch* **have** $\Gamma\vdash\langle c2, Normal\ s'\ \rangle \Rightarrow\notin Fault\ `\ F$
        by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
      with ∗ **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

7.9 Miscellaneous

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

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

lemma *wf-terminates-while*:
 wf $\{(t,s).\ \Gamma\vdash(While\ b\ c)\downarrow Normal\ s \wedge s\in b \wedge$

$\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow Normal\ t\}$
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*: $\Gamma|_M\vdash\ c \downarrow s$
assumes *not-Stuck*: $\Gamma|_M\vdash\langle c,s\ \rangle \Rightarrow\notin\{Stuck\}$
shows $\Gamma\vdash\ c \downarrow s$
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*: $\Gamma|_M\vdash\langle Seq\ c1\ c2,Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
hence *c1-notStuck*: $\Gamma|_M\vdash\langle c1,Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$
by (*rule notStuck-SeqD1*)
show $\Gamma\vdash Seq\ c1\ c2 \downarrow Normal\ s$
proof (*rule terminates.Seq,safe*)
from *c1-notStuck*
show $\Gamma\vdash c1 \downarrow Normal\ s$
by (*rule Seq.hyps*)
next
fix s'
assume *exec*: $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow s'$
show $\Gamma\vdash c2 \downarrow s'$
proof −
from *exec-to-exec-restrict* [*OF exec*] **obtain** t' **where**
exec-res: $\Gamma|_M\vdash\langle c1,Normal\ s\ \rangle \Rightarrow t'$ **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 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 $\Gamma|_M\vdash\langle While\ b\ c,s'\ \rangle \Rightarrow\notin\{Stuck\}$
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*: $\Gamma|_M\vdash\langle Catch\ c1\ c2,Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
hence *c1-notStuck*: $\Gamma|_M\vdash\langle c1,Normal\ s\ \rangle \Rightarrow\notin\{Stuck\}$
by (*rule notStuck-CatchD1*)
show $\Gamma\vdash Catch\ c1\ c2 \downarrow Normal\ s$
proof (*rule terminates.Catch,safe*)
from *c1-notStuck*
show $\Gamma\vdash c1 \downarrow Normal\ s$
by (*rule Catch.hyps*)
next
fix s'
assume *exec*: $\Gamma\vdash\langle c1,Normal\ s\ \rangle \Rightarrow Abrupt\ s'$
show $\Gamma\vdash c2 \downarrow Normal\ s'$
proof −
from *exec-to-exec-restrict* [*OF exec*] **obtain** t' **where**
exec-res: $\Gamma|_M\vdash\langle c1,Normal\ s\ \rangle \Rightarrow t'$ **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 Γ|M⊢⟨c2,Normal s′ ⟩ ⇒∉{Stuck}
        by (auto dest: notStuck-CatchD2)
      with exec-res t′ Catch.hyps
      show ?thesis
        by auto
    qed
  qed
 qed
qed

end
```

8 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

8.1 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*] ⇒ *bool*
(-⊢ (- →/ -) [81,81,81] 100)
  for Γ::$('s,'p,'f)$ *body*
where

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

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

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

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

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

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

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

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

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

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

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

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

8.2 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

8.3 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**
      $steps\text{-}c_2$: $\Gamma\vdash (c_2,\ Normal\ s') \rightarrow^* (c',\ t')$ **and**
      t: ($case\ t\ of$
          $Abrupt\ x \Rightarrow if\ Normal\ s' = t\ then\ c' = Skip \wedge t' = t$
                      $else\ c' = Throw \wedge t' = Normal\ x$
          $|\ \text{-} \Rightarrow c' = Skip \wedge t' = t$)
      by *auto*
  note $steps\text{-}c_2$
  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 $\Gamma\vdash(c,Normal\ s) \rightarrow^* (Throw,\ Normal\ t)$
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 $\Gamma\vdash(c, Abrupt\ t) \rightarrow^* (Skip,\ Abrupt\ t)$
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 $\Gamma\vdash(c,s) \rightarrow^* (Skip,\ Fault\ f)$
using *exec-impl-steps* [*OF exec*]
by *auto*

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

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

lemma *step-Stuck-end*:
  assumes *step*: $\Gamma\vdash\ (c_1,\ s) \rightarrow (c_1',\ s')$
  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*: $\Gamma\vdash\ (c_1,\ s) \rightarrow (c_1',\ s')$
  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**
$exec\text{-}c_1{}'$: $\Gamma\vdash\ \langle c_1{}',Normal\ x'\rangle \Rightarrow s''$ **and**

```
    exec-c₂: Γ⊢ ⟨c₂,s''⟩ ⇒ t
    by cases
  from Seq.hyps (2) Normal exec-c₁' s-Normal
  have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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 c₁ = Guard f g c and
    fail: x ∉ g
    by auto
  hence Γ⊢ ⟨redex c₁,Normal x⟩ ⇒ Fault f
    by (auto intro: exec.intros)
  from exec-redex-Fault [OF this]
  have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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 (∃ r. redex c₁ = Spec r ∧ (∀ t. (x, t) ∉ r)) ∨
       (∃ p. redex c₁ = Call p ∧ Γ p = None)
    by auto
  moreover
  {
    fix r
    assume redex c₁ = Spec r and (∀ t. (x, t) ∉ r)
    hence Γ⊢ ⟨redex c₁,Normal x⟩ ⇒ Stuck
      by (auto intro: exec.intros)
    from exec-redex-Stuck [OF this]
    have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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 c₁ = Call p and Γ p = None
      hence Γ⊢ ⟨redex c₁,Normal x⟩ ⇒ Stuck
        by (auto intro: exec.intros)
      from exec-redex-Stuck [OF this]
      have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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* c_2 s t) **thus** *?case*
by (*cases s*) (*fastforce intro*: *exec.intros elim*: *exec-elim-cases*)+
next
case (*SeqThrow* c_2 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* c_1 s c_1' s' c_2 t)
have *step*: $\Gamma\vdash (c_1, s) \rightarrow (c_1', s')$ **by** *fact*
have *exec'*: $\Gamma\vdash \langle Catch\ c_1'\ c_2, s'\rangle \Rightarrow t$ **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*]
show *?thesis*
proof (*cases*)
fix s''

    assume *exec-c₁'*: Γ⊢ ⟨c₁',Normal x'⟩ ⇒ Abrupt s''
    assume *exec-c₂*: Γ⊢ ⟨c₂,Normal s''⟩ ⇒ t
    from *Catch.hyps (2) Normal exec-c₁' s-Normal*
    have Γ⊢ ⟨c₁,Normal x⟩ ⇒ Abrupt s''
      by *simp*
    from *exec.CatchMatch [OF this exec-c₂] s-Normal*
    show *?thesis* **by** *simp*
  next
    assume *exec-c₁'*: Γ⊢ ⟨c₁',Normal x'⟩ ⇒ t
    assume *t*: ¬ isAbr t
    from *Catch.hyps (2) Normal exec-c₁' s-Normal*
    have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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-c₁: redex c₁ = Guard f g c **and**
    fail: x ∉ g
    by *auto*
  hence Γ⊢ ⟨redex c₁,Normal x⟩ ⇒ Fault f
    by (*auto intro: exec.intros*)
  from *exec-redex-Fault [OF this]*
  have Γ⊢ ⟨c₁,Normal x⟩ ⇒ 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 (∃r. redex c₁ = Spec r ∧ (∀t. (x, t) ∉ r)) ∨
        (∃p. redex c₁ = Call p ∧ Γ p = None)

by *auto*
moreover
{
fix r
assume *redex* c_1 = *Spec* r **and** $(\forall t.\ (x,\ t) \notin r)$
hence $\Gamma\vdash \langle \mathit{redex}\ c_1{,}\mathit{Normal}\ x\rangle \Rightarrow \mathit{Stuck}$
by (*auto intro*: *exec.intros*)
from *exec-redex-Stuck* [*OF this*]
have $\Gamma\vdash \langle c_1{,}\mathit{Normal}\ x\rangle \Rightarrow \mathit{Stuck}$**.**
moreover from *Stuck exec'* **have** $t{=}\mathit{Stuck}$
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** $\Gamma\ p = \mathit{None}$
hence $\Gamma\vdash \langle \mathit{redex}\ c_1{,}\mathit{Normal}\ x\rangle \Rightarrow \mathit{Stuck}$
by (*auto intro*: *exec.intros*)
from *exec-redex-Stuck* [*OF this*]
have $\Gamma\vdash \langle c_1{,}\mathit{Normal}\ x\rangle \Rightarrow \mathit{Stuck}$**.**
moreover from *Stuck exec'* **have** $t{=}\mathit{Stuck}$
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 $s'{=}\mathit{Abrupt}\ x$**.**
with *exec'*
have $t{=}\mathit{Abrupt}\ x$
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 $s'{=}\mathit{Fault}\ f$**.**
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

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

definition *inf*:: ($'s,'p,'f$) *body* $\Rightarrow$ ($'s,'p,'f$) *config* $\Rightarrow$ *bool*
(-$\vdash$ - $\rightarrow \ldots'(\infty')$ [*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*)

8.5 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* 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 *CatchThrow*
thus *?case*
by (*fastforce intro*: *terminates.intros exec.intros*
elim: *terminates-Normal-elim-cases*)
next
case (*CatchSkip* c_2 *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*: $\Gamma \vdash (c,s) \rightarrow^* (c',s')$
shows $\Gamma \vdash c \downarrow s \Longrightarrow \Gamma \vdash c' \downarrow s'$
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*: $\Gamma \vdash (c,s) \rightarrow^+ (c',s')$
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*:

$\forall\, i{<}k.\ \neg\ \mathit{final}\ (\mathit{head}\ (f\ i))$
apply *clarify*
apply (*subgoal-tac* $i < \mathit{Suc}\ k$)
apply *blast*
apply *simp*
done

from *Suc.hyps* [*OF this*]
have *hyp*: $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (\mathit{Seq}\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash\ \mathit{head}\ (f\ i) \to \mathit{head}\ (f\ (i + 1))$.
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 $c'\ \mathit{fs}'\ L'\ s'$ **where** *f-k*: $f\ k = (\mathit{Seq}\ c'\ c_2,\ s')$
by (*cases k*) *auto*
from *inf-comp* [*rule-format*, *of k*] *f-k*
have $\Gamma\vdash(\mathit{Seq}\ c'\ c_2,\ s') \to f\ (k + 1)$
by *simp*
moreover
from *not-fin-Suc* [*rule-format*, *of k*] *f-k*
have $\neg\ \mathit{final}\ (c',s')$
by (*simp add*: *final-def head-def head-com-def*)
ultimately
obtain $c''\ s''$ **where**
$\Gamma\vdash(c',\ s') \to (c'',\ s'')$ **and**
$f\ (k + 1) = (\mathit{Seq}\ c''\ c_2,\ s'')$
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*: $\forall\, i{::}\mathit{nat}.\ \Gamma\vdash f\ i \to f\ (i{+}1)$
assumes *f-0*: $f\ 0 = (\mathit{Catch}\ c_1\ c_2,s)$
assumes *not-fin*: $\forall\, i{<}k.\ \neg\ \mathit{final}\ (\mathit{head}\ (f\ i))$
shows $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (\mathit{Catch}\ c'\ c_2,\ s')) \wedge$
$\Gamma\vdash\mathit{head}\ (f\ i) \to \mathit{head}\ (f\ (i{+}1))$
(**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:
    ∀ i<Suc k. ¬ final (head (f i)) by fact
  from this[rule-format] have not-fin-k:
    ∀ i<k. ¬ final (head (f i))
    apply clarify
    apply (subgoal-tac i < Suc k)
    apply blast
    apply simp
    done

  from Suc.hyps [OF this]
  have hyp: ∀ i<k. (∃ c′ s′. f (i + 1) = (Catch c′ c2, s′)) ∧
                  Γ⊢ head (f i) → 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′ c2, s′)
        by (cases k) auto
      from inf-comp [rule-format, of k] f-k
      have Γ⊢(Catch c′ c2, s′) → f (k + 1)
        by simp
      moreover
      from not-fin-Suc [rule-format, of k] f-k
      have ¬ final (c′,s′)
        by (simp add: final-def head-def head-com-def)
      ultimately
      obtain c′′ s′′ where
         Γ⊢(c′, s′) → (c′′, s′′) and
         f (k + 1) = (Catch c′′ c2, 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*]: $\forall\, i{::}\mathit{nat}.\ \Gamma\vdash f\ i \rightarrow f\ (i{+}\mathit{1})$ **and**
    f-0: $f\ \mathit{0} = (\mathit{Throw},\ s)$
    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 $\Gamma\vdash(c_1,s) \rightarrow \ldots(\infty)\ \vee$
        $(\exists\, s'.\ \Gamma\vdash(c_1,s) \rightarrow^*\ (\mathit{Skip},s') \wedge \Gamma\vdash(c_2,s') \rightarrow \ldots(\infty))$
proof −
  from *inf-comp* **obtain** f **where**
    step: $\forall\, i{::}\mathit{nat}.\ \Gamma\vdash f\ i \rightarrow f\ (i{+}\mathit{1})$ **and**
    f-0: $f\ \mathit{0} = (\mathit{Seq}\ c_1\ c_2,\ s)$
    by (*auto simp add*: *inf-def*)
  from *f-0* **have** *head-f-0*: $\mathit{head}\ (f\ \mathit{0}) = (c_1,s)$
    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*: $\forall\, i{<}k.\ \Gamma\vdash\ \mathit{head}\ (f\ i) \rightarrow \mathit{head}\ (f\ (i\ +\ \mathit{1}))$ **and**
        conf: $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i\ +\ \mathit{1}) = (\mathit{Seq}\ c'\ c_2,\ s'))$
      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* $k\ -\ \mathit{1}$]
    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*: $\Gamma\vdash$*head* (*f 0*) $\rightarrow^*$ *head* (*f k*)
proof (*induct k*)
  case *0* **thus** *?case* **by** *simp*
next
  case (*Suc m*)
  have *step*: $\forall$ *i*<*Suc m*. $\Gamma\vdash$ *head* (*f i*) $\rightarrow$ *head* (*f* (*i* + *1*)) **by** *fact*
  hence $\forall$ *i*<*m*. $\Gamma\vdash$ *head* (*f i*) $\rightarrow$ *head* (*f* (*i* + *1*))
    by *auto*
  hence $\Gamma\vdash$ *head* (*f 0*) $\rightarrow^*$ *head* (*f m*)
    by (*rule Suc.hyps*)
  also from *step* [*rule-format*, *of m*]
  have $\Gamma\vdash$ *head* (*f m*) $\rightarrow$ *head* (*f* (*m* + *1*)) **by** *simp*
  finally show *?case* **by** *simp*
qed
{
  assume *f-k*: *f k* = (*Seq Skip* c_2, s')
  with *steps-head*
  have $\Gamma\vdash$(c_1,*s*) $\rightarrow^*$ (*Skip*,s')
    using *head-f-0*
    by (*simp add*: *head-def head-com-def*)
  moreover
  from *step* [*rule-format*, *of k*] *f-k*
  obtain $\Gamma\vdash$(*Seq 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'=*Normal x* **and** *f-k*: *f k* = (*Seq Throw* c_2, s')
  from *step* [*rule-format*, *of k*] *f-k* s'
  obtain $\Gamma\vdash$(*Seq Throw* c_2,s') $\rightarrow$ (*Throw*,s') **and**
   f-Suc-k: *f* (*k* + *1*) = (*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 $(simp\ add: g\text{-}def)$
from *step*
have $\forall i.\ \Gamma\vdash g\ i \to g\ (i + 1)$
by $(simp\ add: g\text{-}def)$
with *g-0* **have** $\Gamma\vdash(Throw,s') \to \ldots(\infty)$
by $(auto\ simp\ add: inf\text{-}def)$
with *no-inf-Throw*
have *?thesis*
by *auto*
}
ultimately
show *?thesis*
by $(auto\ simp\ add: final\text{-}def\ head\text{-}def\ head\text{-}com\text{-}def)$
next
case *False*
then have *not-fin*: $\forall i.\ \neg\ final\ (head\ (f\ i))$
by *blast*
have $\forall i.\ \Gamma\vdash head\ (f\ i) \to head\ (f\ (i + 1))$
proof
fix k
from *not-fin*
have $\forall i<(Suc\ k).\ \neg\ final\ (head\ (f\ i))$
by *simp*

from *infinite-computation-extract-head-Seq* $[OF\ step\ f\text{-}0\ this\ ]$
show $\Gamma\vdash\ head\ (f\ k) \to head\ (f\ (k + 1))$ **by** *simp*
qed
with *head-f-0* **have** $\Gamma\vdash(c_1,s) \to \ldots(\infty)$
by $(auto\ simp\ add: inf\text{-}def)$
thus *?thesis*
by *simp*
qed
qed

lemma *split-inf-Catch*:
assumes *inf-comp*: $\Gamma\vdash(Catch\ c_1\ c_2,s) \to \ldots(\infty)$
shows $\Gamma\vdash(c_1,s) \to \ldots(\infty)\ \vee$
$(\exists s'.\ \Gamma\vdash(c_1,s) \to^*\ (Throw,Normal\ s') \wedge \Gamma\vdash(c_2,Normal\ s') \to \ldots(\infty))$
proof $-$
from *inf-comp* **obtain** f **where**
step: $\forall i{::}nat.\ \Gamma\vdash f\ i \to f\ (i{+}1)$ **and**
f-0: $f\ 0 = (Catch\ c_1\ c_2,\ s)$
by $(auto\ simp\ add: inf\text{-}def)$
from *f-0* **have** *head-f-0*: $head\ (f\ 0) = (c_1,s)$
by $(simp\ add: head\text{-}def\ head\text{-}com\text{-}def)$
show *?thesis*
proof $(cases\ \exists i.\ final\ (head\ (f\ i)))$
case *True*
define k **where** $k = (LEAST\ i.\ final\ (head\ (f\ i)))$

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-Catch* [*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: $\forall\, i{<}k.\ (\exists\, c'\ s'.\ f\ (i + 1) = (\mathit{Catch}\ c'\ c_2,\ s'))$
  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*: $f\ k = (\mathit{Catch}\ c'\ c_2, s')$
  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*: $f\ k = (\mathit{Catch}\ \mathit{Skip}\ c_2,\ s')$
  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 $\Gamma\vdash(\mathit{Catch}\ \mathit{Skip}\ c_2,s') \rightarrow (\mathit{Skip},s')$ **and**
   f-Suc-k: $f\ (k + 1) = (\mathit{Skip},s')$
    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 c₂, s′)
      with steps-head
      have Γ⊢(c₁,s) →* (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 Γ⊢(Catch Throw c₂,s′) → (c₂,s′) and
        f-Suc-k: f (k + 1) = (c₂,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 = (c₂,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⊢(c₂,s′) → ...(∞)
        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: ∀ 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-Catch [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 Skip-no-step: Γ⊢(Skip,s) → cfg ⟹ P
  apply (erule no-step-final')
  apply (simp add: final-def)
  done

lemma not-inf-Stuck: ¬ Γ⊢(c,Stuck) → ...(∞)
proof (induct c)
  case Skip
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Skip, 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: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Basic g, 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: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Spec r, 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 c1 c2)
  show ?case
  proof
    assume Γ⊢ (Seq c1 c2, Stuck) → ...(∞)
    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: ⋀i. Γ⊢f i → 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: ⋀i. Γ⊢f i → 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 c1 c2)
  show ?case
  proof
    assume Γ⊢ (Catch c1 c2, Stuck) → ...(∞)
    from split-inf-Catch [OF this] Catch.hyps
    show False
      by (auto dest: steps-Stuck-prop)
  qed
qed

lemma not-inf-Fault: ¬ Γ⊢(c,Fault x) → ...(∞)
proof (induct c)
  case Skip
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → 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: ⋀i. Γ⊢f i → 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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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 $(Catch\ c_1\ c_2)$
show *?case*
proof
assume $\Gamma \vdash\ (Catch\ c_1\ c_2,\ Abrupt\ s) \rightarrow \ldots(\infty)$
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*: $\Gamma \vdash c \downarrow s$
shows $\neg\ \Gamma \vdash (c,s) \rightarrow \ldots(\infty)$
using *termi*
proof (*induct*)
case (*Skip s*) **thus** *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = (Skip,\ Normal\ s)$
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*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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*) $\rightarrow \ldots(\infty)$
    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*) $\rightarrow \ldots(\infty)$ **by** *fact*
  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 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*) $\rightarrow \ldots(\infty)$
      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*) $\rightarrow \ldots(\infty)$ **by** *fact*
  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 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*) $\rightarrow \ldots(\infty)$
      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*: $s \in b$ **by** *fact*
have *hyp-c*: $\neg\ \Gamma\vdash$ (*c*, *Normal s*) $\to \ldots(\infty)$ **by** *fact*
have *hyp-w*: $\forall s'.\ \Gamma\vdash \langle c, Normal\ s\rangle \Rightarrow s' \longrightarrow$
$\Gamma\vdash$ *While b c* $\downarrow s' \wedge \neg\ \Gamma\vdash$ (*While b c*, s') $\to \ldots(\infty)$ **by** *fact*
have *not-inf-Seq*: $\neg\ \Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\to \ldots(\infty)$
proof
assume $\Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\to \ldots(\infty)$
from *split-inf-Seq* [*OF this*] *hyp-c hyp-w* **show** *False*
by (*auto intro*: *steps-Skip-impl-exec*)
qed
show *?case*
proof
assume $\Gamma\vdash$ (*While b c*, *Normal s*) $\to \ldots(\infty)$
then obtain *f* **where**
f-step: $\bigwedge i.\ \Gamma\vdash f\ i \to f\ (Suc\ i)$ **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 $\Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\to \ldots(\infty)$
apply (*simp add*: *inf-def*)
apply (*rule-tac* $x=\lambda i.\ f\ (Suc\ i)$ **in** *exI*)
by *simp*
with *not-inf-Seq* **show** *False* **by** *simp*
qed
next
case (*WhileFalse s b c*)
have *b*: $s \notin b$ **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*: $f\ 0 =$ (*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*: $\Gamma\ p =$ *Some bdy* **by** *fact*
have *hyp*: $\neg\ \Gamma\vdash$ (*bdy*, *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*: $f\ 0 =$ (*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 $\Gamma\vdash\ (bdy,\ Normal\ s) \to \ldots(\infty)$
  apply (*simp add*: *inf-def*)
  apply (*rule-tac* $x{=}\lambda i.\ f\ (Suc\ i)$ **in** *exI*)
  by *simp*
with *hyp* **show** *False* **by** *simp*
qed
next
case (*CallUndefined p s*)
have *no-bdy*: $\Gamma\ p = None$ **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*: $f\ 0 = (Call\ p,\ Normal\ s)$
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*: $\neg\ \Gamma\vdash\ (c\ s,\ Normal\ s) \to \ldots(\infty)$ **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*: $f\ 0 = (DynCom\ c,\ Normal\ s)$
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** $\Gamma\vdash\ (c\ s,\ Normal\ s) \to \ldots(\infty)$
  apply (*simp add*: *inf-def*)
  apply (*rule-tac* $x{=}\lambda i.\ f\ (Suc\ i)$ **in** *exI*)
  by *simp*
with *hyp*
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*) $\rightarrow \ldots(\infty)$
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 =
$\{((t,q),(s,p)).\ \Gamma\vdash Call\ p\downarrow Normal\ s \wedge$
$(\exists\, c.\ \Gamma\vdash(Call\ p,Normal\ s) \rightarrow^{+} (c,Normal\ t) \wedge redex\ c = Call\ q)\}$

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 $\Gamma\vdash(redex\ c,s) \to (r',s') \Longrightarrow \Gamma\vdash(c,s) \to (subst\text{-}redex\ c\ r',s')$
by (*induct c*) (*auto intro*: *step.Seq step.Catch*)

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

lemma *steps-redex*:
  assumes *steps*: $\Gamma\vdash\ (r,\ s) \to^*\ (r',\ s')$
  shows $\bigwedge c.\ \Gamma\vdash(subst\text{-}redex\ c\ r,s) \to^*\ (subst\text{-}redex\ c\ r',s')$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl*
  show $\Gamma\vdash\ (subst\text{-}redex\ c\ r',\ s') \to^*\ (subst\text{-}redex\ c\ r',\ s')$
    by *simp*
next
  case (*Trans r s r'' s''*)
  have $\Gamma\vdash\ (r,\ s) \to (r'',\ s'')$ **by** *fact*
  from *step-redex* [*OF this*]
  have $\Gamma\vdash\ (subst\text{-}redex\ c\ r,\ s) \to (subst\text{-}redex\ c\ r'',\ s'')$**.**
  also
  have $\Gamma\vdash\ (subst\text{-}redex\ c\ r'',\ s'') \to^*\ (subst\text{-}redex\ c\ r',\ s')$ **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*: $\Gamma\vdash\ (r,\ s) \to^+\ (r',\ s')$
  shows $\bigwedge c.\ \Gamma\vdash(subst\text{-}redex\ c\ r,s) \to^+\ (subst\text{-}redex\ c\ r',s')$
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*,*case-names Step Trans*])
  case (*Step r' s'*)
  have $\Gamma\vdash\ (r,\ s) \to (r',\ s')$ **by** *fact*
  then have $\Gamma\vdash\ (subst\text{-}redex\ c\ r,\ s) \to (subst\text{-}redex\ c\ r',\ s')$
    by (*rule step-redex*)
  then show $\Gamma\vdash\ (subst\text{-}redex\ c\ r,\ s) \to^+\ (subst\text{-}redex\ c\ r',\ s')$**..**
next
  case (*Trans r' s' r'' s''*)
  have $\Gamma\vdash\ (subst\text{-}redex\ c\ r,\ s) \to^+\ (subst\text{-}redex\ c\ r',\ s')$ **by** *fact*
  also
  have $\Gamma\vdash\ (r',\ s') \to (r'',\ s'')$ **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** $((\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i)\ i,\ f\ i) \in r$
      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** $x{=}\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i$],*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 Γ⊢(c, s) →* f 0
              by (simp add: f0)
          next
            case (Suc n)
```

have $\Gamma\vdash(c,\ s) \rightarrow^* f\ n$ **by** *fact*
with *seq* **show** $\Gamma\vdash(c,\ s) \rightarrow^* f\ (Suc\ n)$
by (*blast intro*: *tranclp-into-rtranclp rtranclp-trans*)
qed
}
hence $\Gamma\vdash(c,s) \rightarrow^* f\ i$
by *iprover*
with *seq* **have**
$(f\ (Suc\ i),\ f\ i) \in \{(y,\ x).\ \Gamma\vdash(c,\ s) \rightarrow^* x \wedge \Gamma\vdash x \rightarrow^+ y\}$
by *clarsimp*
moreover
have $\forall y.\ \Gamma\vdash f\ i \rightarrow^+ y \longrightarrow \Gamma\vdash(c,\ s) \rightarrow^* f\ i \longrightarrow \Gamma\vdash(c,\ s) \rightarrow^* y$
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*: $\forall i.\ (\lambda(t,\ q)\ (s,\ p).$
$\Gamma\vdash Call\ p \downarrow Normal\ s \wedge$
$(\exists c.\ \Gamma\vdash\ (Call\ p,\ Normal\ s) \rightarrow^+ (c,\ Normal\ t) \wedge redex\ c = Call\ q))$
$(f\ (Suc\ i))\ (f\ i)$
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'*: $\forall i.\ \Gamma\vdash Call\ (p\ i) \downarrow Normal\ (s\ i) \wedge$
$(\exists c.\ \Gamma\vdash\ (Call\ (p\ i),\ Normal\ (s\ i)) \rightarrow^+ (c,\ Normal\ (s\ (i+1))) \wedge$
$redex\ c = Call\ (p\ (i+1)))$
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 $\exists c.\ \forall i.\ \Gamma\vdash Call\ (p\ i) \downarrow Normal\ (s\ i) \wedge$
$\Gamma\vdash\ (Call\ (p\ i),\ Normal\ (s\ i)) \rightarrow^+ (c\ i,\ Normal\ (s\ (i+1))) \wedge$
$redex\ (c\ i) = Call\ (p\ (i+1))$
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 ∀ i. Γ⊢(c, s) →* f i ∧ Γ⊢f i → f (Suc i)
  hence ∃f. f 0 = (c, s) ∧ (∀ i. Γ⊢f i → 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) $\to$ (*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) $\to$ (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) $\to^*$ (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 $\Gamma\vdash c_2 \downarrow s'$
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 Γ⊢(Seq c (While b c)) ↓ 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*: $\bigwedge c'\ s'.\ \Gamma\vdash (Call\ p,\ Normal\ s) \to (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  show *?case*
  proof (*cases* Γ *p*)
    case *None*
    thus *?thesis*
      by (*auto intro*: *terminates.intros*)
  next
    case (*Some bdy*)
    then have $\Gamma\vdash (Call\ p,\ Normal\ s) \to (bdy,\ Normal\ s)$
      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*: $\bigwedge c'\ s'.\ \Gamma\vdash (DynCom\ c,\ Normal\ s) \to (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  have $\Gamma\vdash (DynCom\ c,\ Normal\ s) \to (c\ s,\ Normal\ s)$
    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*: $\bigwedge c'\ s'.\ \Gamma\vdash (Guard\ f\ g\ c,\ Normal\ s) \to (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  show *?case*
  proof (*cases* $s \in g$)
    case *True*
    then have $\Gamma\vdash (Guard\ f\ g\ c,\ Normal\ s) \to (c,\ Normal\ s)$
      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*: $\bigwedge c'\ s'.\ \Gamma\vdash (Catch\ c_1\ c_2,\ Normal\ s) \to (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **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: $\Gamma\vdash$ (c_1, *Normal* s) $\rightarrow$ (c'', s'') **and**
        rest: $\Gamma\vdash$ (c'', s'') $\rightarrow^*$ (*Throw*, *Normal* s')
        by (*auto simp add*: *final-def*)
      from *step.Catch* [*OF first*]

have $\Gamma\vdash$ (*Catch* c_1 c_2, *Normal s*) $\rightarrow$ (*Catch* c'' c_2, s'').
from *hyp* [*OF this*]
have $\Gamma\vdash$*Catch* c'' $c_2 \downarrow s''$.
moreover
from *steps-Throw-impl-exec* [*OF rest*]
have $\Gamma\vdash \langle c'',s''\rangle \Rightarrow$ *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* $\{(cfg2,cfg1).\ \Gamma \vdash (c,s) \rightarrow^* cfg1 \wedge \Gamma \vdash cfg1 \rightarrow cfg2\}$
shows $\bigwedge c1\ s1.\ [\![\Gamma \vdash (c,s) \rightarrow^* cfg1;\ \ cfg1=(c1,s1)]\!] \Longrightarrow \Gamma\vdash c1\downarrow s1$
using *wf*
proof (*induct cfg1*, *simp*)
fix *c1 s1*
assume *reach*: $\Gamma\vdash (c,\ s) \rightarrow^* (c1,\ s1)$
assume *hyp-raw*: $\bigwedge y\ c2\ s2.$
$[\![\Gamma\vdash (c1,\ s1) \rightarrow (c2,\ s2);\ \Gamma\vdash (c,\ s) \rightarrow^* (c2,\ s2);\ y = (c2,\ s2)]\!]$
$\Longrightarrow \Gamma\vdash c2 \downarrow s2$
have *hyp*: $\bigwedge c2\ s2.\ \Gamma\vdash (c1,\ s1) \rightarrow (c2,\ s2) \Longrightarrow \Gamma\vdash c2 \downarrow s2$
apply –
apply (*rule hyp-raw*)
apply *assumption*
using *reach*
apply *simp*
apply (*rule refl*)
done

show $\Gamma\vdash c1 \downarrow s1$
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*: $\neg\ \Gamma\vdash (c,\ s) \rightarrow \ldots(\infty)$
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

8.6 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} ∪ *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} ∪ *redexes* c_1

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

lemma *redex-in-redexes*: *redex c* ∈ *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 $\bigwedge r\ r'.\ [\![\Gamma\vdash(r,s) \to (r',s');\ r \in redexes\ c]\!]$

$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \to (c',s') \wedge r' \in \mathit{redexes}\ c'$
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*: $\Gamma\vdash (r,\ s) \to (r',\ s')$ **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: $\Gamma\vdash (c_1,\ s) \to (c',\ s')$ **and**
      r': $r' \in \mathit{redexes}\ c'$
      by *blast*
    from *step.Seq* [*OF step-*c_1]
    have $\Gamma\vdash (\mathit{Seq}\ c_1\ c_2,\ s) \to (\mathit{Seq}\ c'\ c_2,\ s')$**.**
    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* c_1 c_2)
  have $r \in$ *redexes* (*Catch* c_1 c_2) **by** *fact*
  hence r: r = *Catch* c_1 c_2 $\vee$ $r \in$ *redexes* c_1
    by *simp*
  have *step-r*: $\Gamma\vdash$ $(r,\ s) \rightarrow (r',\ s')$ **by** *fact*
  from r **show** *?case*
  proof
    assume r = *Catch* c_1 c_2
    with *step-r*
    show *?case*
      by (*auto simp add*: *root-in-redexes*)
  next
    assume r: $r \in$ *redexes* c_1
    from *Catch.hyps* (*1*) [*OF step-r this*]
    obtain c' **where**
      step-c_1: $\Gamma\vdash$ $(c_1,\ s) \rightarrow (c',\ s')$ **and**
      r': $r' \in$ *redexes* c'
      by *blast*
    from *step.Catch* [*OF step*-c_1]
    have $\Gamma\vdash$ (*Catch* c_1 c_2, s) $\rightarrow$ (*Catch* c' c_2, s')**.**
    with r'
    show *?case*
      by *auto*
  qed
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*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
  case *Refl*
  then
  show $\exists\, c'.\ \Gamma\vdash\ (c,\ s') \rightarrow^* (c',\ s') \wedge r' \in$ *redexes* c'
    by *auto*
next
  case (*Trans* r s r'' s'')
  have $\Gamma\vdash$ $(r,\ s) \rightarrow (r'',\ s'')$ $r \in$ *redexes* c **by** *fact+*
  from *step-redexes* [*OF this*]
  obtain c' **where**
    step: $\Gamma\vdash$ $(c,\ s) \rightarrow (c',\ s'')$ **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 $\exists c'.\ \Gamma\vdash(c,s) \rightarrow (c',s') \wedge$ *Seq* r' $c_2 \in$ *redexes* c'
proof −
from *step.Seq* [*OF step*]
have $\Gamma\vdash$ (*Seq* r c_2, s) $\rightarrow$ (*Seq* r' c_2, s').

from *step-redexes* [*OF this Seq*]
show *?thesis* **.**
qed

lemma *steps-redexes-Seq*:
assumes *steps*: $\Gamma\vdash$ $(r,\ s) \rightarrow^* (r',\ s')$
shows $\bigwedge c.$ *Seq* $r\ c_2 \in$ *redexes* $c \Longrightarrow$
$\exists\, c'.\ \Gamma\vdash(c,s) \rightarrow^* (c',s') \wedge$ *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 $\Gamma\vdash$ $(r,\ s) \rightarrow (r'',\ s'')$ *Seq* $r\ c_2 \in$ *redexes* c **by** *fact+*
from *step-redexes-Seq* [*OF this*]
obtain c' **where**
step: $\Gamma\vdash$ $(c,\ s) \rightarrow (c',\ s'')$ **and**
r'': *Seq* $r''\ c_2 \in$ *redexes* c'
by *blast*
note *step*
also
from *Trans.hyps* (*3*) [*OF* r'']
obtain c'' **where**
steps: $\Gamma\vdash$ $(c',\ s'') \rightarrow^* (c'',\ s')$ **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*: $\Gamma\vdash$ $(r,\ s) \rightarrow^+ (r',\ s')$
shows $\bigwedge c.$ *Seq* $r\ c_2 \in$ *redexes* c
$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \rightarrow^+ (c',s') \wedge$ *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 $\Gamma\vdash$ $(r,\ s) \rightarrow (r',\ s')$ *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

9 Hoare Logic for Total Correctness

theory *HoareTotalDef* **imports** *HoarePartialDef Termination* **begin**

9.1 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*$] \Rightarrow$ *bool*
(-$\models_{t'/\text{-}}$/ - - -,- [61,60,1000, 20, 1000,1000] 60)
where
$\Gamma\models_{t/F}$ P c Q,A $\equiv$ $\Gamma\models_{/F}$ P c Q,A $\wedge$ $(\forall s \in$ *Normal* ‘ P. $\Gamma\vdash c\downarrow s)$

definition
cvalidt::
$[('s,'p,'f)$ *body*,$('s,'p)$ *quadruple set*,$'f$ *set*,
$'s$ *assn*,$('s,'p,'f)$ *com*,$'s$ *assn*,$'s$ *assn*$] \Rightarrow$ *bool*
(-,-$\models_{t'/\text{-}}$/ - - -,- [61,60, 60,1000, 20, 1000,1000] 60)
where
$\Gamma,\Theta\models_{t/F}$ P c Q,A $\equiv$ $(\forall (P,p,Q,A)\in\Theta.$ $\Gamma\models_{t/F}$ P (*Call* p) Q,A) $\longrightarrow$ 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)

9.2 Properties of Validity

lemma *validtI*:
$[\![\bigwedge s\ t.\ [\![\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t;s \in P;t \notin Fault\ `\ F]\!] \Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A;$
$\bigwedge s.\ s \in P \Longrightarrow \Gamma\vdash\ c\downarrow(Normal\ s)\ ]\!]$
$\Longrightarrow \Gamma\models_{t/F}\ P\ c\ Q,A$
by (*auto simp add*: *validt-def valid-def*)

lemma *cvalidtI*:
$[\![\bigwedge s\ t.\ [\![\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A;\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t;s \in P;$
$t \notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A;$
$\bigwedge s.\ [\![\forall (P,p,Q,A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A;\ s\in P]\!] \Longrightarrow\ \Gamma\vdash c\downarrow(Normal\ s)]\!]$
$\Longrightarrow \Gamma,\Theta\models_{t/F}\ P\ c\ Q,A$
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*)

9.3 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*:
$[\![(P,p,Q,A) \in Specs;$
$wf\ r;$
$Specs\text{-}wf = (\lambda p\ \sigma.\ (\lambda(P,q,Q,A).\ (P \cap \{s.\ ((s,q),(\sigma,p)) \in r\},q,Q,A))\ \text{`}\ Specs);$
$\forall (P,p,Q,A)\in Specs.$
$p \in dom\ \Gamma \wedge (\forall \sigma.\ \Gamma,\Theta \cup Specs\text{-}wf\ p\ \sigma\vdash_{t/F} (\{\sigma\} \cap P)\ (the\ (\Gamma\ p))\ Q,A)$
$]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (Call\ p)\ Q,A$

| *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 τ
$\subseteq \Theta' \cup$ *Specs-wf* p τ
by *blast*
have $\forall(P,p,Q,A)\in$*Specs*. $p \in$ *dom* $\Gamma \wedge$
($\forall\tau.\ \Gamma,\Theta \cup$ *Specs-wf* p $\tau\vdash_{t/F}$ ($\{\tau\} \cap P$) (*the* (Γ p)) Q,A $\wedge$
($\forall x.\ \Theta \cup$ *Specs-wf* p τ
$\subseteq x \longrightarrow$
$\Gamma,x\vdash_{t/F}$ ($\{\tau\} \cap P$) (*the* (Γ p)) Q,A)) **by** *fact*
hence $\forall(P,p,Q,A)\in$*Specs*. $p \in$ *dom* $\Gamma \wedge$

$(\forall \tau.\ \Gamma,\Theta' \cup$ *Specs-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-wf p* τ **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* Θ *F c Q A* Θ')
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* Θ *F P c Q A* Θ')
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*)+

9.4 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

10 Properties of Total Correctness Hoare Logic

theory *HoareTotalProps* **imports** *SmallStep HoareTotalDef HoarePartialProps* **begin**

10.1 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 $\Gamma\vdash\langle \mathit{Spec}\ r\ ,\mathit{Normal}\ s\rangle \Rightarrow t$
        $s \in \{s.\ (\forall t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,\ t) \in r)\}$
    thus $t \in \mathit{Normal}$ ` $Q \cup \mathit{Abrupt}$ ` 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*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q{,}A$
assume *P*: $s \in P$
thus $\Gamma{\vdash}\mathit{Cond}\ b\ c1\ c2 \downarrow \mathit{Normal}\ s$
using *cvalidt-termD* [*OF valid-c1 ctxt*] *cvalidt-termD* [*OF valid-c2 ctxt*]
by ($\mathit{cases}\ s \in b$) (*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 $\Gamma{,}\Theta \models_{t/F} P\ (\mathit{While}\ b\ c)\ (P \cap -\ b){,}A$
proof (*rule cvalidtI*)
fix $s\ t$
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models_{t/F}} P\ (\mathit{Call}\ p)\ Q{,}A$
assume *wprems*: $\Gamma{\vdash}\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ s\rangle \Rightarrow t\ s \in P\ t \notin \mathit{Fault}\ `\ F$
from *wf*
have $\bigwedge t.\ [\![\Gamma{\vdash}\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ s\rangle \Rightarrow t;\ s \in P;\ t \notin \mathit{Fault}\ `\ F]\!]$
$\Longrightarrow t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}\ `\ A$
proof (*induct*)
fix $s\ t$
assume *hyp*:
$\bigwedge s'\ t.\ [\![(s'{,}s){\in}r;\ \Gamma{\vdash}\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ s'\rangle \Rightarrow t;\ s' \in P;\ t \notin \mathit{Fault}\ `\ F]\!]$
$\Longrightarrow t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}\ `\ A$
assume *exec*: $\Gamma{\vdash}\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ s\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin \mathit{Fault}\ `\ F$
from *exec*
show $t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}\ `\ A$
proof (*cases*)
fix s'
assume *b*: $s{\in}b$
assume *exec-c*: $\Gamma{\vdash}\langle c{,}\mathit{Normal}\ s\rangle \Rightarrow s'$
assume *exec-w*: $\Gamma{\vdash}\langle \mathit{While}\ b\ c{,}s'\rangle \Rightarrow t$
from *exec-w t-notin-F* **have** $s' \notin \mathit{Fault}\ `\ F$
by (*auto dest*: *Fault-end*)
from *exec-c P b valid-c ctxt this*
have *s'*: $s' \in \mathit{Normal}\ `\ (\{s'.\ (s',\ s) \in r\} \cap P) \cup \mathit{Abrupt}\ `\ A$
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∉b t=Normal s with P show ?thesis by simp
  qed
 qed
 with wprems show t ∈ Normal ‘ (P ∩ − b) ∪ Abrupt ‘ A by blast
next
 fix s
 assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ⊨t/F P (Call p) Q,A
 assume s ∈ P
 with wf
 show Γ⊢While b c ↓ Normal s
 proof (induct)
  fix s
  assume hyp: ⋀s'. ⟦(s',s)∈r; s' ∈ P⟧
                ⟹ Γ⊢While b c ↓ Normal s'
  assume P: s ∈ P
  show Γ⊢While b c ↓ Normal s
  proof (cases s ∈ b)
    case False with P show ?thesis
      by (blast intro: terminates.intros)
  next
    case True
    with valid-c P ctxt
    have Γ⊢c ↓ Normal s
      by (simp add: cvalidt-def validt-def)
    moreover
    {
      fix s'
      assume exec-c: Γ⊢⟨c,Normal s⟩ ⇒ s'
      have Γ⊢While b c ↓ s'
      proof (cases s')
        case (Normal s'')
```

```
      with exec-c P True valid-c ctxt
      have s': s' ∈ Normal ` ({s'. (s', s) ∈ r} ∩ 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: Γ,Θ ⊨t/F (g ∩ P) c Q,A by fact
 show Γ,Θ ⊨t/F (g ∩ P) Guard f g 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⊢⟨Guard f g c,Normal s⟩ ⇒ t
  assume t-notin-F: t ∉ Fault ` F
  assume P:s ∈ (g ∩ P)
  from exec P have Γ⊢⟨c,Normal s⟩ ⇒ t
   by cases auto
  from valid-c ctxt this P t-notin-F
  show t ∈ Normal ` Q ∪ Abrupt ` A
   by (rule cvalidt-postD)
 next
  fix s
  assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ ⊨t/F P (Call p) Q,A
  assume P:s ∈ (g ∩ P)
  thus Γ⊢Guard f g c ↓ 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: Γ,Θ ⊨t/F (g ∩ P) c Q,A by fact
 have f-F: f ∈ F by fact
 show Γ,Θ ⊨t/F P Guard f g 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⊢⟨Guard f g c,Normal s⟩ ⇒ t
  assume t-notin-F: t ∉ Fault ` F
  assume P:s ∈ P
  from exec f-F t-notin-F have g: s ∈ g
   by cases auto
```

    with P **have** P': $s \in g \cap P$
      by *blast*
    from *exec g* **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 $P{:}s \in P$
    thus $\Gamma\vdash Guard\ f\ g\ c\ \downarrow Normal\ s$
      by (*auto intro*: *terminates.intros cvalidt-termD* [*OF valid-c ctxt*])
  qed
next
  case (*CallRec P p Q A Specs r Specs-wf* Θ *F*)
  have p: $(P{,}p{,}Q{,}A) \in Specs$ **by** *fact*
  have *wf*: *wf r* **by** *fact*
  have *Specs-wf*:
   $Specs\text{-}wf = (\lambda p\ \tau.\ (\lambda(P{,}q{,}Q{,}A).\ (P \cap \{s.\ ((s,\ q){,}\tau{,}p) \in r\}{,}q{,}Q{,}A))\ `\ Specs)$ **by**
fact
  from *CallRec.hyps*
  have *valid-body*:
   $\forall (P,\ p,\ Q,\ A){\in}Specs.\ p \in dom\ \Gamma \wedge$
      $(\forall \tau.\ \Gamma{,}\Theta \cup Specs\text{-}wf\ p\ \tau{\models_{t/F}}\ (\{\tau\} \cap P)\ the\ (\Gamma\ p)\ Q{,}A)$ **by** *auto*
  show $\Gamma{,}\Theta \models_{t/F} P\ (Call\ p)\ Q{,}A$
  proof −
   {
    fix τp
    assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma{\models_{t/F}} P\ (Call\ p)\ Q{,}A$
    from *wf*
    have $\bigwedge\tau\ p\ P\ Q\ A.\ \ [\![\tau p = (\tau{,}p);\ (P{,}p{,}Q{,}A) \in Specs]\!] \Longrightarrow$
           $\Gamma{\models_{t/F}} (\{\tau\} \cap P)\ (the\ (\Gamma\ (p)))\ Q{,}A$
    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: $(P,\ p,\ Q,\ A) \in Specs$ **by** *fact*
      {
        fix $q\ P'\ Q'\ A'$
        assume q: $(P'{,}q{,}Q'{,}A') \in Specs$
        have $\Gamma{\models_{t/F}} (P' \cap \{s.\ ((s{,}q),\ \tau{,}p) \in r\})\ (Call\ q)\ Q'{,}A'$
        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:
        Γ⊨t/F ({s} ∩ P′) the (Γ q) Q′,A′
        by − (rule WF.hyps, auto)
      from Pre q
      have Pre′: s ∈ {s} ∩ P′
        by auto
      from exec-q show t ∈ Normal ‘ Q′ ∪ Abrupt ‘ A′
      proof (cases)
        fix bdy
        assume bdy: Γ q = Some bdy
        assume exec-bdy: Γ⊢⟨bdy,Normal s⟩ ⇒ t
        from valid-bdy [simplified bdy option.sel]  t-notin-F exec-bdy Pre′
        have t ∈ Normal ‘ Q′ ∪ Abrupt ‘ A′
          by (auto simp add: validt-def valid-def)
        with Pre q
        show ?thesis
          by auto
      next
        assume Γ q = None
        with q valid-body have False by auto
        thus ?thesis ..
      qed
    next
      fix s
      assume Pre: s ∈ P′ ∩ {s. ((s,q), τ,p) ∈ r}
      from Pre q τp
      have valid-bdy:
        Γ ⊨t/F ({s} ∩ P′) (the (Γ q)) Q′,A′
        by − (rule WF.hyps, auto)
      from Pre q
      have Pre′: s ∈ {s} ∩ P′
        by auto
      from valid-bdy ctxt Pre′
      have Γ⊢the (Γ q) ↓ Normal s
        by (auto simp add: validt-def)
      with valid-body q
      show Γ⊢Call q↓ Normal s
        by (fastforce intro: terminates.Call)
    qed
  }
  hence ∀(P, p, Q, A)∈Specs-wf p τ. Γ⊨t/F P Call p Q,A
    by (auto simp add: cvalidt-def Specs-wf)
  with ctxt have ∀(P, p, Q, A)∈Θ ∪ Specs-wf p τ. Γ⊨t/F P Call p Q,A
    by auto
  with p valid-body
  show Γ ⊨t/F ({τ} ∩ P) (the (Γ p)) Q,A
    by (simp add: cvalidt-def) blast
qed
```

}
note *lem* = *this*
have *valid-body'*:
  $\bigwedge\tau.\ \forall(P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (Call\ p)\ Q,A \Longrightarrow$
  $\forall(P,p,Q,A)\in Specs.\ \Gamma\models_{t/F}\ (\{\tau\}\cap P)\ (the\ (\Gamma\ p))\ Q,A$
  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*: $\Gamma\ p = Some\ bdy$
    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 $\Gamma\ p = None$
    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 $\Gamma\vdash(the\ (\Gamma\ p)) \downarrow Normal\ s$
      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*: $\forall s\in P.\ \Gamma,\Theta\models_{t/F} P\ (c\ s)\ Q,A$ **by** *simp*
  show $\Gamma,\Theta\models_{t/F} P$ *DynCom 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*: $\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow t$
    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*)
      assume $\Gamma\vdash\langle c\ s,Normal\ s\rangle \Rightarrow t$
      from *cvalidt-postD* [*OF valid-c* [*rule-format*, *OF P*] *ctxt this P t-notin-F*]
      show *?thesis* **.**
    qed
  next
    fix *s*
    assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
    assume *P*: $s \in P$
    show $\Gamma\vdash DynCom\ c \downarrow Normal\ s$
    proof −
      from *cvalidt-termD* [*OF valid-c* [*rule-format*, *OF P*] *ctxt P*]
      have $\Gamma\vdash c\ s \downarrow Normal\ s$ **.**
      thus *?thesis*
        by (*rule terminates.intros*)
    qed
  qed
next
  case (*Throw* Θ *F A Q*)
  show $\Gamma,\Theta \models_{t/F} A$ *Throw Q*,*A*
  proof (*rule cvalidtI*)
    fix *s t*
    assume $\Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow t$ $s \in A$
    then show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
      by *cases simp*
  next
    fix *s*
    show $\Gamma\vdash Throw \downarrow Normal\ s$
      by (*rule terminates.intros*)
  qed
next
  case (*Catch* Θ *F P* c_1 *Q R* c_2 *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 $(Conseq\ P\ \Theta\ F\ c\ Q\ A)$
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'*:
Γ,{}$\vdash_{t/F}$ 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*: Γ,{}$\vdash_{t/F}$ P c Q,A **shows** Γ,{}$\vdash_{/F}$ P c Q,A
proof −
from *total* **have** Γ,{}$\models_{t/F}$ 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

10.2 Completeness

lemma *MGT-valid*:
$\Gamma\models_{t/F}$ {*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*}
proof (*rule validtI*)
fix *s t*
assume Γ⊢⟨*c*,*Normal s*⟩ ⇒ *t*
s ∈ {*s*. *s* = *Z* ∧ Γ⊢⟨*c*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧ Γ⊢*c*↓*Normal* *s*}
t ∉ *Fault* ‘ *F*
thus *t* ∈ *Normal* ‘ {*t*. Γ⊢⟨*c*,*Normal Z*⟩ ⇒ *Normal t*} ∪
Abrupt ‘ {*t*. Γ⊢⟨*c*,*Normal Z*⟩ ⇒ *Abrupt t*}
apply (*cases t*)
apply (*auto simp add*: *final-notin-def*)
done
next
fix *s*
assume *s* ∈ {*s*. *s*=*Z* ∧ Γ⊢⟨*c*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧ Γ⊢*c*↓*Normal* *s*}
thus Γ⊢*c*↓*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* ∧ Γ⊢⟨*Skip*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*Skip* ↓ *Normal s*}
Skip
{*t*. Γ⊢⟨*Skip*,*Normal Z*⟩ ⇒ *Normal t*},{*t*. Γ⊢⟨*Skip*,*Normal Z*⟩ ⇒ *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* ∧ Γ⊢⟨*Basic f*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*Basic f* ↓ *Normal s*}
Basic f
{*t*. Γ⊢⟨*Basic f*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Basic f*,*Normal Z*⟩ ⇒ *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* ∧ Γ⊢⟨*Spec r*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*Spec r* ↓ *Normal s*}
Spec r
{*t*. Γ⊢⟨*Spec r*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Spec r*,*Normal Z*⟩ ⇒ *Abrupt t*}
apply (*rule hoaret.Spec* [*THEN conseqPre*])
apply (*clarsimp simp add*: *final-notin-def*)
apply (*case-tac* ∃ *t*. (*Z*,*t*) ∈ *r*)
apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
done
next
case (*Seq c1 c2*)
have *hyp-c1*: ∀ *Z*. $\Gamma,\Theta\vdash_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*c1*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*c1*↓*Normal s*}
c1
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Abrupt t*}
using *Seq.hyps* **by** *iprover*
have *hyp-c2*: ∀ *Z*. $\Gamma,\Theta\vdash_{t/F}$ {*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 *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 $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash c1{\downarrow}Normal\ s\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
hence $\Gamma,\Theta \vdash_{t/F}$ $(\{s.\ s{=}Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash(Cond\ b\ c1\ c2){\downarrow}Normal\ s\}{\cap}b)$
c1

$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondTrue*
elim: *terminates-Normal-elim-cases*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash c2\downarrow Normal\ s\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{t/F} (\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash(Cond\ b\ c1\ c2)\downarrow Normal\ s\}\cap-b)$
$c2$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondFalse*
elim: *terminates-Normal-elim-cases*)
ultimately
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash(Cond\ b\ c1\ c2)\downarrow Normal\ s\}$
$(Cond\ b\ c1\ c2)$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 $?P' = \lambda Z.\ \{t.\ (Z,t)\in ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash(While\ b\ c)\downarrow Normal\ t\}$
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 $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash(While\ b\ c)\downarrow Normal\ s\}$
$(While\ b\ c)$
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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*) (*While b c*) (*?P′ Z* ∩ − *b*),(*?A Z*)
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix *Z*
from *While*
have *hyp-c*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash c\downarrow Normal\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$ **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 σ s
assume $s\in \{\sigma\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash(While\ b\ c)\downarrow Normal\ t\}$
$\cap\ b$
then obtain
s-eq-σ: $s{=}\sigma$ **and**
Z-s-unroll: $(Z,s) \in\ ?unroll$ **and**
noabort: $\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)$ **and**
while-term: $\Gamma\vdash(While\ b\ c)\downarrow Normal\ s$ **and**
s-in-b: $s\in b$
by *blast*
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash c\downarrow Normal\ t\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Normal\ t\} \longrightarrow$
$t \in \{t.\ (t,\sigma) \in\ ?r\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow\ e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash(While\ b\ c)\downarrow Normal\ t\})\ \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Abrupt\ t\} \longrightarrow$
$t \in \{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\})$
(**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 $(Z,e) \in$ *?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*: $[\![e\in b;\Gamma\vdash\langle While\ b\ c, Normal\ r\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))]\!]$
$\Longrightarrow$ *?Prop r e*
assume *Z-r*:
$(Z,\ r) \in \{(Z,\ r).\ Z \in b \wedge \Gamma\vdash\langle c, Normal\ Z\rangle \Rightarrow Normal\ r\}$
with *WhileNoFault*
have $\Gamma\vdash\langle While\ b\ c, Normal\ r\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
from *hyp* [*OF e-in-b this*] **obtain**
cNoFault: $\Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$ **and**
Abrupt-r: $\forall u.\ \Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c, Normal\ r\rangle \Rightarrow Abrupt\ u$
by *simp*

{
fix *u* **assume** $\Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow Abrupt\ u$
with *Abrupt-r* **have** $\Gamma\vdash\langle While\ b\ c, Normal\ r\rangle \Rightarrow Abrupt\ u$ **by** *simp*
moreover from *Z-r* **obtain**
$Z \in b$ $\ \Gamma\vdash\langle c, Normal\ Z\rangle \Rightarrow Normal\ r$
by *simp*
ultimately have $\Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ u$
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)
 from noStuck-Call
 have ∀ s ∈ {s. s=Z ∧ Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                  Γ⊢Call p↓ 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*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z\ \wedge\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash c\downarrow Normal\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

using *Guard* **by** *iprover*
show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Guard\ f\ g\ c\downarrow\ Normal\ s\}$
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*
from *hyp-c*
have $\Gamma,\Theta\vdash_{t/F}$ $(g \cap \{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\}\cup Fault\ `\ (-F))\wedge$
$\Gamma\vdash Guard\ f\ g\ c\downarrow\ Normal\ s\})$
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*
elim: *terminates-Normal-elim-cases*)
done
from *True this*
show *?thesis*
by (*rule conseqPre* [*OF Guarantee*]) *auto*
next
case *False*
from *hyp-c*
have $\Gamma,\Theta\vdash_{t/F}$ $(g \cap \{s.\ s \in g \wedge s{=}Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\}\cup Fault\ `\ (-F))\wedge$
$\Gamma\vdash Guard\ f\ g\ c\downarrow\ Normal\ s\}\ )$
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 *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 $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$

$\wedge$

$\Gamma\vdash \mathit{Throw} \downarrow \mathit{Normal}\ s\}$
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 $\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$

$\Gamma\vdash c_1 \downarrow \mathit{Normal}\ s\}$
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 $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F)) \wedge$

$\Gamma\vdash \mathit{Catch}\ c_1\ c_2 \downarrow \mathit{Normal}\ s\}$
c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t \wedge \Gamma\vdash c_2 \downarrow \mathit{Normal}\ t \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros terminates.intros*
elim: *terminates-Normal-elim-cases*
simp add: *final-notin-def*)
moreover
have
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash c_2 \downarrow \mathit{Normal}\ s\}\ c_2$
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{t/F} \{s.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ s \wedge \Gamma\vdash c_2 \downarrow \mathit{Normal}\ s \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
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 $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F)) \wedge$

$\Gamma\vdash \mathit{Catch}\ c_1\ c_2 \downarrow \mathit{Normal}\ s\}$
$\mathit{Catch}\ c_1\ c_2$
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$

by (*rule 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 $\Gamma,\Theta{\vdash}_{t/F}\ \{s.\ s{=}Z \wedge \Gamma{\vdash}\langle Spec\ r,Normal\ s\rangle \Rightarrow{\notin}(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma{\vdash}Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma{\vdash}(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge$
$Spec\ r \in redexes\ c')\}$
$Spec\ r$
$\{t.\ \Gamma{\vdash}\langle Spec\ r,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma{\vdash}\langle Spec\ r,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoaret.Spec* [*THEN conseqPre*])

```
    apply (clarsimp)
    apply (case-tac ∃ t. (Z,t) ∈ r)
    apply (auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros)
    done
next
  case (Seq c1 c2)
  have hyp-c1:
    ∀ Z. Γ,Θ⊢t/F {s. s=Z ∧ Γ⊢⟨c1,Normal s⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                   Γ⊢Call p↓Normal σ ∧
                 (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal s) ∧ c1 ∈ redexes c')}
                c1
               {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Abrupt t}
    using Seq.hyps by iprover
  have hyp-c2:
    ∀ Z. Γ,Θ⊢t/F {s. s=Z ∧ Γ⊢⟨c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                  Γ⊢Call p↓Normal σ ∧
                 (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal s) ∧ c2 ∈ redexes c')}
                c2
               {t. Γ⊢⟨c2,Normal Z⟩ ⇒ Normal t},
               {t. Γ⊢⟨c2,Normal Z⟩ ⇒ Abrupt t}
    using Seq.hyps (2) by iprover
  have c1: Γ,Θ⊢t/F {s. s=Z ∧ Γ⊢⟨Seq c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault `
(−F)) ∧
                  Γ⊢Call p↓Normal σ ∧
             (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal s) ∧
                  Seq c1 c2 ∈ redexes c')}
               c1
               {t. Γ⊢⟨c1,Normal Z⟩ ⇒ Normal t ∧
                   Γ⊢⟨c2,Normal t⟩ ⇒∉({Stuck} ∪ Fault ` (−F)) ∧
                   Γ⊢Call p↓Normal σ ∧
                  (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal t) ∧
                        c2 ∈ redexes c')},
               {t. Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒ Abrupt t}
  proof (rule ConseqMGT [OF hyp-c1],clarify,safe)
    assume Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ` (−F))
    thus Γ⊢⟨c1,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ` (−F))
      by (blast dest: Seq-NoFaultStuckD1)
  next
    fix c'
    assume steps-c': Γ⊢ (Call p, Normal σ) →+ (c', Normal Z)
    assume red: Seq c1 c2 ∈ redexes c'
    from redexes-subset [OF red] steps-c'
    show ∃ c'. Γ⊢ (Call p, Normal σ) →+ (c', Normal Z) ∧ c1 ∈ redexes c'
      by (auto iff: root-in-redexes)
  next
    fix t
    assume Γ⊢⟨Seq c1 c2,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ` (−F))
           Γ⊢⟨c1,Normal Z⟩ ⇒ 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'*: $\Gamma\vdash$ $(Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
assume *red*: $Seq\ c1\ c2 \in redexes\ c'$
assume *exec-c1*: $\Gamma\vdash \langle c1,Normal\ Z\rangle \Rightarrow Normal\ t$
show $\exists\, c'.\ \Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ t) \wedge c2 \in redexes\ c'$
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 $\Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma)\rightarrow^{+}(c',Normal\ s) \wedge While\ b\ c \in redexes\ c')\}$
$(While\ b\ c)$
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F} (?P'\ Z)\ (While\ b\ c)\ (?P'\ Z \cap -\ b),(?A\ Z)$
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix Z
from *While*
have *hyp-c*: $\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

$\Gamma\vdash$*Call p*$\downarrow$*Normal* σ $\wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^{+} (c',\mathit{Normal}\ s) \wedge c \in \mathit{redexes}\ c')\}$
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*: $s\in \{\tau\} \cap$
$\{t.\ (Z, t) \in\ ?unroll\ \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash$*Call p*$\downarrow$ *Normal* σ $\wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^{+}$
$(c',\mathit{Normal}\ t) \wedge \mathit{While}\ b\ c \in \mathit{redexes}\ c')\}$
$\cap\ b$
then obtain c' **where**
s-eq-τ: $s=\tau$ **and**
Z-s-unroll: $(Z,s) \in\ ?unroll$ **and**
noabort:$\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)$ **and**
termi: $\Gamma\vdash$*Call p* $\downarrow$ *Normal* σ **and**
reach: $\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^{+} (c',\mathit{Normal}\ s)$ **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: $\Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^{+} (c'',\mathit{Normal}\ s)$
$\mathit{Seq}\ c\ (\mathit{While}\ b\ c) \in \mathit{redexes}\ c''$
proof −
note *reach*
also from *s-in-b*
have $\Gamma\vdash(\mathit{While}\ b\ c,\mathit{Normal}\ s) \to (\mathit{Seq}\ c\ (\mathit{While}\ b\ c),\mathit{Normal}\ s)$
by (*rule step.WhileTrue*)
from *step-redexes* [*OF this red-c'*] **obtain** c'' **where**
step: $\Gamma\vdash (c', \mathit{Normal}\ s) \to (c'', \mathit{Normal}\ s)$ **and**
red-c'': $\mathit{Seq}\ c\ (\mathit{While}\ b\ c) \in \mathit{redexes}\ c''$
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 $s \in \{t.\ t = s \wedge \Gamma \vdash \langle c, Normal\ t\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ t) \wedge c \in redexes\ c')\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow Normal\ t\} \longrightarrow$
$t \in \{t.\ (t,\tau) \in ?r\} \cap$
$\{t.\ (Z,\ t) \in ?unroll \wedge$
$(\forall e.\ (Z,e) \in ?unroll \longrightarrow e \in b$
$\longrightarrow \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ t) \wedge$
$While\ b\ c \in redexes\ c')\}) \wedge$
$(\forall t.\ t \in \{t.\ \Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow Abrupt\ t\} \longrightarrow$
$t \in \{t.\ \Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ t\})$
  (**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*: $\Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow Normal\ t$
    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'': $\Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c'', Normal\ t)$
      $(While\ b\ c) \in redexes\ c''$
    proof −
      note *reach-c* (*1*)
      also from *s-in-b*
      have $\Gamma \vdash (While\ b\ c, Normal\ s) \rightarrow (Seq\ c\ (While\ b\ c), Normal\ s)$
        by (*rule step.WhileTrue*)
      have $\Gamma \vdash (Seq\ c\ (While\ b\ c),\ Normal\ s) \rightarrow^{+}$
      $(While\ b\ c,\ Normal\ t)$

```
          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:  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⊢Call p↓Normal σ ∧
                 (∃ c'. Γ⊢(Call p,Normal σ) →⁺ (c',Normal s) ∧
                     While b c ∈ redexes c')}
  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 $\forall\, t.\ t{\in}(?P'\ s \cap -\ b)$
$\longrightarrow t{\in}\{t.\ \Gamma{\vdash}\langle \mathit{While}\ b\ c{,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$
by *blast*
next
from P **show** $\forall\, t.\ t{\in}?A\ s \longrightarrow t{\in}?A\ Z$
by *simp*
qed
qed
next
case $(\mathit{Call}\ q)$
let $?P = \{s.\ s{=}Z \wedge \Gamma{\vdash}\langle \mathit{Call}\ q\ {,}\mathit{Normal}\ s\rangle \Rightarrow{\notin}(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ ({-}F)) \wedge$
$\Gamma{\vdash}\mathit{Call}\ p{\downarrow}\mathit{Normal}\ \sigma \wedge$
$(\exists\, c'.\ \Gamma{\vdash}(\mathit{Call}\ p{,}\mathit{Normal}\ \sigma) \rightarrow^{+} (c'{,}\mathit{Normal}\ s) \wedge \mathit{Call}\ q \in \mathit{redexes}\ c')\}$
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**
$\forall\, q{\in}\mathit{dom}\ \Gamma.\ \forall\, Z.$
$\Gamma{,}\Theta{\vdash}_{t/F}\{s.\ s{=}Z \wedge \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ s\rangle \Rightarrow{\notin}(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ ({-}F)) \wedge$
$\Gamma{\vdash}\mathit{Call}\ q{\downarrow}\mathit{Normal}\ s \wedge ((s{,}q){,}(\sigma{,}p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}{,}$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*simp add*: *exec-Call-body'* *noFaultStuck-Call-body'* [*simplified*]
terminates-Normal-Call-body)
from *Call-hyp q-defined* **have** *Call-hyp'*:
$\forall\, Z.\ \Gamma{,}\Theta \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ s\rangle \Rightarrow{\notin}(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ ({-}F)) \wedge$
$\Gamma{\vdash}\mathit{Call}\ q{\downarrow}\mathit{Normal}\ s \wedge ((s{,}q){,}(\sigma{,}p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}{,}$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q{,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by *auto*
show
$\Gamma{,}\Theta{\vdash}_{t/F}\ ?P$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q\ {,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}{,}$
$\{t.\ \Gamma{\vdash}\langle \mathit{Call}\ q\ {,}\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [*OF Call-hyp'*]*,safe*)
fix c'
assume *termi*: $\Gamma{\vdash}\mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma$
assume *steps-c'*: $\Gamma{\vdash}\ (\mathit{Call}\ p{,}\ \mathit{Normal}\ \sigma) \rightarrow^{+} (c'{,}\ \mathit{Normal}\ Z)$
assume *red-c'*: $\mathit{Call}\ q \in \mathit{redexes}\ c'$
show $\Gamma{\vdash}\mathit{Call}\ q \downarrow \mathit{Normal}\ Z$
proof $-$
from *steps-preserves-termination'* [*OF steps-c' termi*]
have $\Gamma{\vdash}c' \downarrow \mathit{Normal}\ Z$ **.**

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'*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \to^+ (c',\ Normal\ Z)$
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*:
$\bigwedge s'.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s = Z \wedge \Gamma\vdash\langle c\ s',Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \to^+ (c',Normal\ s) \wedge c\ s' \in redexes\ c')\}$
$(c\ s')$
$\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \to^+ (c',Normal\ s) \wedge DynCom\ c \in redexes$
$c')\}$
$(c\ Z)$
$\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow$
$Abrupt\ t\}$
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*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \to^+ (c',\ Normal\ Z)$
assume c': $DynCom\ c \in redexes\ c'$
have $\Gamma\vdash\ (DynCom\ c,\ Normal\ Z) \to (c\ Z,Normal\ Z)$
by (*rule step.DynCom*)
from *step-redexes* [*OF this c'*] **obtain** c'' **where**
step: $\Gamma\vdash\ (c',\ Normal\ Z) \to (c'',\ Normal\ Z)$ **and** c'': $c\ Z \in redexes\ c''$
by *blast*
note *steps* **also note** *step*
finally show $\exists c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \to^+ (c',\ Normal\ Z) \wedge c\ Z \in redexes$

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*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge c \in redexes\ c')\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Guard.hyps* **by** *iprover*
show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge Guard\ f\ g\ c \in redexes$
$c')\}$
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* σ) $\rightarrow^+$ (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* σ) $\rightarrow^+$ (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* σ $\wedge$
($\exists\, c'$. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^+$ (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* σ) $\rightarrow^+$ (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*:
$\forall\, Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}\ Z \wedge \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$

$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^+ (c',\mathit{Normal}\ s) \wedge$
$c_1 \in \mathit{redexes}\ c')\}$
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*
have *hyp-c2*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s= Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F)) \wedge$
$\Gamma\vdash\mathit{Call}\ p\downarrow\ \mathit{Normal}\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^+ (c',\mathit{Normal}\ s) \wedge c_2 \in \mathit{redexes}\ c')\}$
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*
have
$\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
$\wedge$
$\Gamma\vdash\mathit{Call}\ p\downarrow\ \mathit{Normal}\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma)\to^+(c',\mathit{Normal}\ s) \wedge$
$\mathit{Catch}\ c_1\ c_2 \in \mathit{redexes}\ c')\}$
c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\text{`}(-F)) \wedge \Gamma\vdash\mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma$
$\wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \to^+ (c',\mathit{Normal}\ t) \wedge c_2 \in \mathit{redexes}\ c')\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*clarify*,*safe*)
assume $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
thus $\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash\ (\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \to^+ (c',\ \mathit{Normal}\ Z)$
assume c': $\mathit{Catch}\ c_1\ c_2 \in \mathit{redexes}\ c'$
from *steps redexes-subset* [*OF this*]
show $\exists\, c'.\ \Gamma\vdash\ (\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \to^+ (c',\ \mathit{Normal}\ Z) \wedge c_1 \in \mathit{redexes}\ c'$
by (*auto iff*: *root-in-redexes*)
next
fix t
assume $\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
thus $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
by (*auto intro*: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
$\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
thus $\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix $c'\ t$
assume *steps-c'*: $\Gamma\vdash\ (\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \to^+ (c',\ \mathit{Normal}\ Z)$

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 $\exists\, c'.\ \Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ t) \land c_2 \in redexes\ c'$
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 $\Gamma,\Theta\vdash_{t/F} \{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t\ \land$
$\Gamma\vdash\langle c_2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\ \land$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma\ \land$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ t) \land c_2 \in redexes\ c')\}$
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*:
$\forall\, q \in dom\ \Gamma.\ \forall\, Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s{=}Z \land \Gamma\vdash\langle Call\ q,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\ \land$
$\Gamma\vdash Call\ q{\downarrow}Normal\ s \land ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$

$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 σ*
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-wf* $p\ \sigma = (\lambda(P,q,Q,A).$

$(P \cap \{s.\ ((s,q),\sigma,p) \in \textit{termi-call-steps}\ \Gamma\},\ q,\ Q,\ A))$ ' *Specs* **for** $p\ \sigma$

have Γ,*Specs-wf* $p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle \textit{the}\ (\Gamma\ p),\textit{Normal}\ s\rangle \Rightarrow\notin(\{\textit{Stuck}\} \cup \textit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \textit{the}\ (\Gamma\ p)\downarrow\textit{Normal}\ s\})$
$(\textit{the}\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle \textit{the}\ (\Gamma\ p),\textit{Normal}\ Z\rangle \Rightarrow \textit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \textit{the}\ (\Gamma\ p),\textit{Normal}\ Z\rangle \Rightarrow \textit{Abrupt}\ t\}$
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 Γ,*Specs-wf* $p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ s\rangle \Rightarrow\notin(\{\textit{Stuck}\} \cup \textit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \textit{Call}\ p\downarrow\textit{Normal}\ s\})$
$(\textit{the}\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ Z\rangle \Rightarrow \textit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ Z\rangle \Rightarrow \textit{Abrupt}\ t\}$
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*=$(\bigcup p{\in}\textit{dom}\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ s\rangle \Rightarrow\notin(\{\textit{Stuck}\} \cup \textit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \textit{Call}\ p\downarrow\textit{Normal}\ s\}$,
p,
$\{t.\ \Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ Z\rangle \Rightarrow \textit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \textit{Call}\ p,\textit{Normal}\ Z\rangle \Rightarrow \textit{Abrupt}\ t\})\})$,
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* $\Longrightarrow$ Γ,{}$\vdash_{t/F}$ *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 Γ,{}$\vdash_{t/F}$ *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

10.3 And Now: Some Useful Rules

10.3.1 Modify Return

lemma *ProcModifyReturn-sound*:
  assumes *valid-call*: $\Gamma,\Theta \models_{t/F}$ *P call init p return' c Q*,*A*
  assumes *valid-modif*:
  $\forall\sigma$. $\Gamma,\Theta \models_{/UNIV}$ {σ} (*Call p*) (*Modif* σ),(*ModifAbr* σ)
  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*: 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*: $\Gamma\ p = Some\ bdy$
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 Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Stuck*
t = *Stuck*
with *bdy* **have** Γ⊢⟨*call init p return′ c* ,*Normal s*⟩ ⇒ *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 Γ⊢⟨*call init p return′ c* ,*Normal s*⟩ ⇒ *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*: ∀ (*P*, *p*, *Q*, *A*)∈Θ. Γ $\models_{t/F}$ *P* (*Call p*) *Q*,*A*
hence ∀ (*P*, *p*, *Q*, *A*)∈Θ. Γ $\models_{/F}$ *P* (*Call p*) *Q*,*A*
by (*auto simp add*: *validt-def*)
then have *ctxt′*: ∀ (*P*, *p*, *Q*, *A*)∈Θ. Γ $\models_{/UNIV}$ *P* (*Call p*) *Q*,*A*
by (*auto intro*: *valid-augment-Faults*)
assume *P*: *s* ∈ *P*
from *valid-call ctxt P*
have *call*: Γ⊢*call init p return′ c*↓ *Normal s*
by (*rule cvalidt-termD*)
show Γ⊢*call init p return c* ↓ *Normal s*
proof (*cases p* ∈ *dom* Γ)
case *True*
with *call* **obtain** *bdy* **where**
bdy: Γ *p* = *Some bdy* **and** *termi-bdy*: Γ⊢*bdy* ↓ *Normal* (*init s*) **and**
termi-c: ∀ *t*. Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Normal t* ⟶
Γ⊢*c s t* ↓ *Normal* (*return′ s t*)
by *cases auto*
{
fix *t*
assume *exec-bdy*: Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Normal t*
hence Γ⊢*c s t* ↓ *Normal* (*return s t*)
proof −
from *exec-bdy bdy*
have Γ⊢⟨(*Call p*),*Normal* (*init s*)⟩ ⇒ *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 σ*),(*ModifAbr σ*)
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 σ*),(*ModifAbr σ*)
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*: $\Gamma\ p = Some\ bdy$
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 *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* $p \in dom\ \Gamma$)
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 *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* σ),(*ModifAbr* σ)
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

10.3.2 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* 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 $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models_{/F}$ 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*: $\Gamma\vdash\langle$*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'*:
  $\forall\sigma.\ \Gamma,\Theta\models_{/UNIV}\ \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  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*: $\Gamma\ (p\ s) = Some\ bdy$
  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.Call*)
  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*: $\Gamma\ (p\ s) = Some\ bdy$
  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*: $\Gamma\ (p\ s) = Some\ bdy$
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*: Γ⊢*call init* (*p s*) *return′ c*↓ *Normal s*
  by *cases*
have Γ⊢*call init* (*p s*) *return c* ↓ *Normal s*
proof (*cases p s* ∈ *dom* Γ)
  case *True*
  with *call* **obtain** *bdy* **where**
    bdy: Γ (*p s*) = *Some bdy* **and** *termi-bdy*: Γ⊢*bdy* ↓ *Normal* (*init s*) **and**
    termi-c: ∀ *t*. Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Normal t* ⟶
              Γ⊢*c s t* ↓ *Normal* (*return′ s t*)
    by *cases auto*
  {
    fix *t*
    assume *exec-bdy*: Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Normal t*
    hence Γ⊢*c s t* ↓ *Normal* (*return s t*)
    proof −
      from *exec-bdy bdy*
      have Γ⊢⟨*Call* (*p s*),*Normal* (*init s*)⟩ ⇒ *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 Γ⊢*dynCall init p return c* ↓ *Normal s*
  by (*iprover intro*: *terminates-dynCall*)
qed

lemma *dynProcModifyReturn*:
assumes *dyn-call*: Γ,Θ$\vdash_{t/F}$ *P dynCall init p return′ c Q*,*A*
assumes *ret-modif*:
  ∀ *s t*. *t* ∈ *Modif* (*init s*)
     ⟶ *return′ s t* = *return s t*
assumes *ret-modifAbr*: ∀ *s t*. *t* ∈ *ModifAbr* (*init s*)
              ⟶ *return′ s t* = *return s t*
assumes *modif*:
  ∀ *s* ∈ *P*. ∀ σ.
    Γ,Θ$\vdash_{/UNIV}$ {σ} *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
shows Γ,Θ $\vdash_{t/F}$ *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*: $\Gamma\ (p\ s) = Some\ bdy$
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*: $\Gamma\ (p\ s) = Some\ bdy$
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 *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 Γ (*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* Γ)
case *True*
with *call* **obtain** *bdy* **where**
bdy: Γ (*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* σ),(*ModifAbr* σ)
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

10.3.3 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*: $\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow t$
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*: $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A.
then 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 .
with *t* **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

10.3.4 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*: $\Gamma,\Theta \vdash_{/\{\}} P\ (strip\text{-}guards\ (-F)\ c)\ UNIV,UNIV$
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-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-guards c* $\downarrow$ *Normal s***.**
  thus $\Gamma \vdash c \downarrow$ *Normal s*

by (*rule terminates-merge-guards-to-terminates*)
qed

lemma *MergeGuardsD*:
 assumes *deriv*: Γ,$\Theta$$\vdash_{t/F}$ *P merge-guards c Q*,*A*
 shows Γ,$\Theta$$\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*: Γ,$\Theta$$\models_{t/\{\}}$ *P c′ Q*,*A*
 shows Γ,$\Theta$$\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*: Γ,$\Theta$$\vdash_{t/\{\}}$ *P c′ Q*,*A*
 shows Γ,$\Theta$$\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

10.3.5 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 Γ,{}$\vdash_{t/F}$ 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*)

10.3.6 Miscellaneous

lemma *augment-Faults*:
assumes *deriv-c*: Γ,{}$\vdash_{t/F}$ P c Q,A
assumes *F*: $F \subseteq F'$
shows Γ,{}$\vdash_{t/F'}$ 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*: Γ,$\Theta \models_{/F}$ P c Q,A
  shows Γ,$\Theta \models_{t/F}$ P c Q,A
using *termination partial-corr*
by (*auto simp add*: *cvalidt-def validt-def cvalid-def*)

lemma *TerminationPartial*:
  assumes *partial-deriv*: Γ,$\Theta \vdash_{/F}$ P c Q,A
  assumes *termination*: $\forall s \in P.\ \Gamma \vdash c \downarrow Normal\ s$
  shows Γ,$\Theta \vdash_{t/F}$ 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*: Γ,$\Theta \vdash_{/F}$ P c Q,A
  assumes *termination*: $\forall s \in P.\ strip\ F'\ \Gamma \vdash strip\text{-}guards\ F'\ c \downarrow Normal\ s$
  shows Γ,$\Theta \vdash_{t/F}$ 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

11 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*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z)$
$\Longrightarrow$
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ UNIV,UNIV$
apply (*rule allI*)
apply (*erule conseq*)
apply *auto*
done

lemma *TrivPostNoAbr*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ (Q'\ Z),\{\}$
$\Longrightarrow$
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ c\ UNIV,\{\}$

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 $\Gamma,\Theta\vdash_{t/F} P\ \mathit{DynCom}\ c\ Q,A$
  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 $\Gamma,\Theta\vdash_{t/F} P\ (\mathit{specAnno}\ P'\ c\ Q'\ A')\ Q,A$
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'*:
 $[\![P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z \wedge$
  $(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q) \wedge (\forall t.\ t \in A'\ Z \longrightarrow t \in\ A)\};$
  $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z);$
  $\forall Z.\ c\ Z = c\ \mathit{undefined}$
  $]\!] \Longrightarrow$
   $\Gamma,\Theta\vdash_{t/F} P\ (\mathit{specAnno}\ P'\ c\ Q'\ A')\ Q,A$

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*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta{\vdash}_{t/F}\ P\ Spec\ (if\text{-}rel\ b\ f\ g\ h)\ Q,A$
apply (*rule Spec*)
apply (*auto simp add*: *if-rel-def*)
done

lemma *Seq* [*trans*, *intro?*]:
$[\![\Gamma,\Theta{\vdash}_{t/F}\ P\ c_1\ R,A;\ \Gamma,\Theta{\vdash}_{t/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta{\vdash}_{t/F}\ P\ Seq\ c_1\ c_2\ Q,A$
by (*rule hoaret.Seq*)

lemma *SeqSwap*:
$[\![\Gamma,\Theta{\vdash}_{t/F}\ R\ c2\ Q,A;\ \Gamma,\Theta{\vdash}_{t/F}\ P\ c1\ R,A]\!] \Longrightarrow \Gamma,\Theta{\vdash}_{t/F}\ P\ Seq\ c1\ c2\ Q,A$
by (*rule Seq*)

lemma *BSeq*:
$[\![\Gamma,\Theta\vdash_{t/F} P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{t/F} R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (bseq\ c_1\ c_2)\ Q,A$
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 $\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$
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*:
$[\![\Gamma,\Theta\vdash_{t/F} P1\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F} P2\ c2\ Q,A;$
$P \subseteq \{s.\ (s\in b \longrightarrow s\in P1) \wedge (s\notin b \longrightarrow s\in P2)\}]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c1\ c2)\ Q,A$
by (*rule Cond*)

lemma *Cond'*:
$[\![P \subseteq \{s.\ (b \subseteq P1) \wedge (-\ b \subseteq P2)\};\Gamma,\Theta\vdash_{t/F} P1\ c1\ Q,A;\ \Gamma,\Theta\vdash_{t/F} P2\ c2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (Cond\ b\ c1\ c2)\ Q,A$
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*:
 $[\![\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ P \subseteq g \cap R]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ Guard\ f\ g\ c\ Q,A$
  by (*rule Guard*)

lemma *Guarantee*:
 $[\![P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ f \in F]\!]$
  $\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (Guard\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*, *of* - - - - - $\{s.\ s \in g \longrightarrow s \in R\}$])
apply *assumption*
apply (*erule conseqPre*)
apply *auto*

done

lemma *GuaranteeSwap*:
$\llbracket\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (Guard\ f\ g\ c)\ Q,A$
by (*rule Guarantee*)

lemma *GuardStrip*:
$\llbracket P \subseteq R;\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (Guard\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*])
apply *auto*
done

lemma *GuardStripSwap*:
$\llbracket\Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ P \subseteq R;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (Guard\ f\ g\ c)\ Q,A$
by (*rule GuardStrip*)

lemma *GuaranteeStrip*:
$\llbracket P \subseteq R;\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (guaranteeStrip\ f\ g\ c)\ Q,A$
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeStripSwap*:
$\llbracket\Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ P \subseteq R;\ f \in F\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (guaranteeStrip\ f\ g\ c)\ Q,A$
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*:
$\llbracket\ \Gamma,\Theta\vdash_{t/F} R\ c\ Q,A;\ P \subseteq g \cap R\rrbracket$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ guaranteeStrip\ f\ g\ c\ Q,A$
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*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
  $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
  $(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
  assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
  assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ Z)$ *Call* $p\ (Q'\ Z),(A'\ Z)$
  shows $\Gamma,\Theta\vdash_{t/F}\ P$ (*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*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
  assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ Z)$ *Call* $p\ (Q'\ Z),(A'\ Z)$

shows $\Gamma,\Theta\vdash_{t/F}$ 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*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ Call\ p\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{t/F}$ P (*call init p return c*) Q,A
apply (*rule ProcSpec* [*OF - c p*])
using *adapt*
apply *simp*
done

lemma *FCall*:
$\Gamma,\Theta\vdash_{t/F}$ P (*call init p return* ($\lambda s\ t.\ c$ (*result t*))) Q,A
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}$ P (*fcall init p return result c*) Q,A
by (*simp add*: *fcall-def*)

lemma *ProcRec*:
assumes *deriv-bodies*:
$\forall p{\in}Procs.$
$\forall \sigma\ Z.\ \Gamma,\Theta\cup(\bigcup q{\in}Procs.\ \bigcup Z.$
$\{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\})$
$\vdash_{t/F} (\{\sigma\} \cap P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *Procs-defined*: $Procs \subseteq dom\ \Gamma$
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*:
$\forall p{\in}Procs.$

$\forall \sigma\ Z.\ \Gamma,\Theta'\ \sigma\ p \vdash_{t/F} (\{\sigma\} \cap P\ p\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *Procs-defined*: $\mathit{Procs} \subseteq \mathit{dom}\ \Gamma$
shows $\forall p \in \mathit{Procs}.\ \forall Z.\ \ \Gamma,\Theta \vdash_{t/F} (P\ p\ Z)\ \mathit{Call}\ p\ (Q\ p\ Z),(A\ p\ Z)$
using *ctxt deriv-bodies*
apply *simp*
apply (*erule ProcRec* [*OF - wf Procs-defined*])
done

lemma *ProcRecList*:
assumes *deriv-bodies*:
$\forall p \in \mathit{set}\ \mathit{Procs}.$
$\forall \sigma\ Z.\ \Gamma,\Theta \cup (\bigcup q \in \mathit{set}\ \mathit{Procs}.\ \bigcup Z.$
$\{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\})$
$\vdash_{t/F} (\{\sigma\} \cap P\ p\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *dist*: *distinct Procs*
assumes *Procs-defined*: $\mathit{set}\ \mathit{Procs} \subseteq \mathit{dom}\ \Gamma$
shows $\forall p \in \mathit{set}\ \mathit{Procs}.\ \forall Z.$
$\Gamma,\Theta \vdash_{t/F} (P\ p\ Z)\ \mathit{Call}\ p\ (Q\ p\ Z),(A\ p\ Z)$
using *deriv-bodies wf Procs-defined*
by (*rule ProcRec*)

lemma *ProcRecSpecs*:
$[\![\forall \sigma.\ \forall (P,p,Q,A) \in \mathit{Specs}.$
$\Gamma,\Theta \cup ((\lambda(P,q,Q,A).\ (P \cap \{s.\ ((s,q),(\sigma,p)) \in r\},q,Q,A))\ \text{`}\ \mathit{Specs})$
$\vdash_{t/F} (\{\sigma\} \cap P)\ (\mathit{the}\ (\Gamma\ p))\ Q,A;$
$\mathit{wf}\ r;$
$\forall (P,p,Q,A) \in \mathit{Specs}.\ p \in \mathit{dom}\ \Gamma]\!]$
$\Longrightarrow \forall (P,p,Q,A) \in \mathit{Specs}.\ \Gamma,\Theta \vdash_{t/F} P\ (\mathit{Call}\ p)\ Q,A$
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 $\forall p{\in}\{p\}.\ \forall Z.\ \Gamma{,}\Theta{\vdash}_{t/F}\ (P\ Z)$ *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 $\forall Z.\ \Gamma{,}\Theta{\vdash}_{t/F}\ (P\ Z)$ *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*: $\Gamma{,}\Theta{\vdash}_{t/F}\ P'$ *body* $Q{,}A$
assumes *body*: $\Gamma\ p = \mathit{Some\ body}$
shows $\Gamma{,}\Theta{\vdash}_{t/F}\ P$ *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 hoaret-augment-context* [*OF deriv-body*])
apply *blast*
apply *fastforce*
done

lemma *CallBody*:
assumes *adapt*: $P \subseteq \{s.\ \mathit{init}\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma{,}\Theta{\vdash}_{t/F}\ (P'\ s)$ *body* $\{t.\ \mathit{return}\ s\ t \in R\ s\ t\}{,}\{t.\ \mathit{return}\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma{,}\Theta{\vdash}_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q{,}A$
assumes *body*: $\Gamma\ p = \mathit{Some\ body}$
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** Γ=Γ, *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* σ),{}
  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* σ),{}
  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* $P'{=}\lambda Z''.\ \{t.\ t{=}Z'' \wedge \mathit{return}\ Z\ t \in R\ Z\ t\}$ **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*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ s\ Z \wedge$
$(\forall t \in Q'\ s\ Z.\ \mathit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ s\ Z.\ \mathit{return}\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s{\in}\ P.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ s\ Z)\ \mathit{Call}\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ \mathit{dynCall}\ \mathit{init}\ p\ \mathit{return}\ c\ Q,A$
proof −
from *adapt* **have** $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ s\ Z \wedge$
$(\forall t.\ t \in Q'\ s\ Z \longrightarrow \mathit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ s\ Z \longrightarrow \mathit{return}\ s\ t \in A)\}$
by *blast*
from *this c p* **show** *?thesis*
by (*rule DynProc*)
qed

lemma *DynProcStaticSpec*:
assumes *adapt*: $P \subseteq \{s.\ s \in S \wedge (\exists Z.\ \mathit{init}\ s \in P'\ Z\ \wedge$

$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
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*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ Call\ q\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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 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*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV}\ \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),\{\}$
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*
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*} ∩ *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*} ∩ *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*:
$[\![\forall Z.\ \Gamma,\{\}\vdash_{t/F} (P\ Z)\ p\ (Q\ Z),(A\ Z)]\!] \Longrightarrow$
$\forall Z.\ \Gamma,\{\}\vdash_{t/UNIV} (P\ Z)\ p\ (Q\ Z),(A\ Z)$
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

11.0.1 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

12 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-,-/$\vdash$ (-/ (-)/ -,/-)) [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-/$\vdash_{'/-}$ (-/ (-)/ -,/-)) [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-/$\vdash$ (-/ (-)/ -,/-)) [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-,-/$\vdash_{'/-}$ (-/ (-)/ -)) [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-,-/$\vdash$ (-/ (-)/ -)) [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-/$\vdash_{'/-}$ (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoarep-emptyCtx-noAbr-emptyFaults::
[('s,'p,'f) *body*,'s *assn*,('s,'p,'f) *com*, 's *assn*] => *bool*
((3-/$\vdash$ (-/ (-)/ -)) [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-/$\vdash_{t'/-}$ (-/ (-)/ -,/-)) [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-,-/$\vdash_{t'/-}$ (-/ (-)/ -)) [61,60,60,1000,20,1000]60)

-hoaret-noAbr-emptyFaults::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-},\text{-}/\vdash_t\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoaret-emptyCtx-noAbr::
$[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/\vdash_{t\ '/\text{-}}\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,60,1000,20,1000]60)$

-hoaret-emptyCtx-noAbr-emptyFaults::
$[('s,'p,'f)\ body,'s\ assn,('s,'p,'f)\ com,\ 's\ assn] => bool$
$((3\text{-}/\vdash_t\ (\text{-}/\ (\text{-})/\ \text{-}))\ [61,1000,20,1000]60)$

syntax (*ASCII*)

-hoarep-emptyFaults::
$[('s,'p,'f)\ body,('s,'p)\ quadruple\ set,$
$'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] \Rightarrow bool$
$((3\text{-},\text{-}/|-\ (\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,60,1000,20,1000,1000]60)$

-hoarep-emptyCtx::
$[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] => bool$
$((3\text{-}/|-'/\text{-}\ (\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,60,1000,20,1000,1000]60)$

-hoarep-emptyCtx-emptyFaults::
$[('s,'p,'f)\ body,'s\ assn,('s,'p,'f)\ com,\ 's\ assn,'s\ assn] => bool$
$((3\text{-}/|-(\text{-}/\ (\text{-})/\ \text{-},/\text{-}))\ [61,1000,20,1000,1000]60)$

-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-,-/|−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-/|−t'/- (-/ (-)/ -,/-)) [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-/|−t(-/ (-)/ -,/-)) [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-,-/|−t'/- (-/ (-)/ -)) [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-,-/|−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-/|−t'/- (-/ (-)/ -)) [61,60,1000,20,1000]60)

-hoaret-emptyCtx-noAbr-emptyFaults::
[('s,'p,'f) body,'s assn,('s,'p,'f) com, 's assn] => bool
((3-/|−t(-/ (-)/ -)) [61,1000,20,1000]60)

translations

$\Gamma\vdash\ P\ c\ Q{,}A\ \ == \Gamma\vdash_{/\{\}}\ P\ c\ Q{,}A$
$\Gamma\vdash_{/F}\ P\ c\ Q{,}A\ \ == \Gamma{,}\{\}\vdash_{/F}\ P\ c\ Q{,}A$

$\Gamma{,}\Theta\vdash\ P\ c\ Q\ \ == \Gamma{,}\Theta\vdash_{/\{\}}\ P\ c\ Q$
$\Gamma{,}\Theta\vdash_{/F}\ P\ c\ Q\ \ == \Gamma{,}\Theta\vdash_{/F}\ P\ c\ Q{,}\{\}$
$\Gamma{,}\Theta\vdash\ P\ c\ Q{,}A == \Gamma{,}\Theta\vdash_{/\{\}}\ P\ c\ Q{,}A$

$\Gamma\vdash\ P\ c\ Q\ \ \ \ == \ \Gamma\vdash_{/\{\}}\ P\ c\ Q$
$\Gamma\vdash_{/F}\ P\ c\ Q\ \ == \Gamma{,}\{\}\vdash_{/F}\ P\ c\ Q$
$\Gamma\vdash_{/F}\ P\ c\ Q\ \ <= \ \Gamma\vdash_{/F}\ P\ c\ Q{,}\{\}$
$\Gamma\vdash\ P\ c\ Q\ \ \ \ <= \ \Gamma\vdash\ P\ c\ Q{,}\{\}$

$\Gamma\vdash_t\ P\ c\ Q{,}A\ \ \ == \Gamma\vdash_{t/\{\}}\ P\ c\ Q{,}A$
$\Gamma\vdash_{t/F}\ P\ c\ Q{,}A\ \ == \Gamma{,}\{\}\vdash_{t/F}\ P\ c\ Q{,}A$

$\Gamma,\Theta\vdash_t\ P\ c\ Q\quad == \Gamma,\Theta\vdash_{t/\{\}}\ P\ c\ Q$
$\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q == \Gamma,\Theta\vdash_{t/F}\ P\ c\ Q,\{\}$
$\Gamma,\Theta\vdash_t\ P\ c\ Q,A\quad == \Gamma,\Theta\vdash_{t/\{\}}\ P\ c\ Q,A$

$\Gamma\vdash_t\ P\ c\ Q\quad == \Gamma\vdash_{t/\{\}}\ P\ c\ Q$
$\Gamma\vdash_{t/F}\ P\ c\ Q\ == \Gamma,\{\}\vdash_{t/F}\ P\ c\ Q$
$\Gamma\vdash_{t/F}\ P\ c\ Q\ <= \ \Gamma\vdash_{t/F}\ P\ c\ Q,\{\}$
$\Gamma\vdash_t\ P\ c\ Q\quad <= \ \Gamma\vdash_t\ P\ c\ Q,\{\}$

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*:: $('a \times 'b)$ *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)) ≤ 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) ∉ range g
            by blast
          hence False by simp
          thus ?thesis
            by simp
        qed
      qed
    qed
  qed
  with k [symmetric] have ∀ i. k ≤ i ⟶ f (g i) = f z
    by simp
  hence ∀ i. k ≤ i ⟶ f (g (Suc i)) = f (g i)
    by simp
  with g have ∀ i. k ≤ i ⟶ (g (Suc i),(g i)) ∈ r
    by (auto simp add: in-measure-iff order-less-le )
  hence ∀ i. (g (Suc (i+k)),(g (i+k))) ∈ r
    by simp
  then
  have ∃ f. ∀ i. (f (Suc i), f i) ∈ r
    by − (rule exI [where x=λi. 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: (∀ x. x = k ⟶ Q) = Q
                       (∀ x. k = x ⟶ Q) = Q
  by auto

end
```

13 State Space Template

```
theory StateSpace imports Hoare
begin
```

record $'g$ *state* = *globals*::$'g$

definition
upd-globals:: $('g \Rightarrow 'g) \Rightarrow ('g,'z)$ *state-scheme* $\Rightarrow ('g,'z)$ *state-scheme*
where
upd-globals upd s = *s*(|*globals* := *upd* (*globals s*)|)

record $('g, 'n, 'val)$ *stateSP* = $'g$ *state* +
locals :: $'n \Rightarrow 'val$

lemma *upd-globals-conv*: *upd-globals f* = (λs. *s*(|*globals* := *f* (*globals s*)|))
by (*rule ext*) (*simp add*: *upd-globals-def*)

end

14 Alternative Small Step Semantics

theory *AlternativeSmallStep* **imports** *HoareTotalDef*
begin

This is the small-step semantics, which is described and used in my PhD-thesis [9]. It decomposes the statement into a list of statements and finally executes the head. So the redex is always the head of the list. The equivalence between termination (based on the big-step semantics) and the absence of infinite computations in this small-step semantics follows the same lines of reasoning as for the new small-step semantics. However, it is technically more involved since the configurations are more complicated. Thats why I switched to the new small-step semantics in the "main trunk". I keep this alternative version and the important proofs in this theory, so that one can compare both approaches.

14.1 Small-Step Computation: $\Gamma\vdash(cs,\ css,\ s) \rightarrow (cs',\ css',\ s')$

type-synonym $('s,'p,'f)$ *continuation* = $('s,'p,'f)$ *com list* $\times$ $('s,'p,'f)$ *com list*

type-synonym $('s,'p,'f)$ *config* =
$('s,'p,'f)$*com list* $\times$ $('s,'p,'f)$*continuation list* $\times$ $('s,'f)$ *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
Skip: $\Gamma\vdash$(*Skip*#*cs*,*css*,*Normal s*) $\rightarrow$ (*cs*,*css*,*Normal s*)
| *Guard*: $s \in g \Longrightarrow \Gamma\vdash$(*Guard f g c*#*cs*,*css*,*Normal s*) $\rightarrow$ (*c*#*cs*,*css*,*Normal s*)

| *GuardFault*: $s \notin g \Longrightarrow \Gamma\vdash(Guard\ f\ g\ c\#cs,css,Normal\ s) \rightarrow (cs,css,Fault\ f)$

| *FaultProp*: $\Gamma\vdash(c\#cs,css,Fault\ f) \rightarrow (cs,css,Fault\ f)$
| *FaultPropBlock*: $\Gamma\vdash([],(nrms,abrs)\#css,Fault\ f) \rightarrow (nrms,css,Fault\ f)$

| *AbruptProp*: $\Gamma\vdash(c\#cs,css,Abrupt\ s) \rightarrow (cs,css,Abrupt\ s)$

| *ExitBlockNormal*:
$\Gamma\vdash([],(nrms,abrs)\#css,Normal\ s) \rightarrow (nrms,css,Normal\ s)$
| *ExitBlockAbrupt*:
$\Gamma\vdash([],(nrms,abrs)\#css,Abrupt\ s) \rightarrow (abrs,css,Normal\ s)$

| *Basic*: $\Gamma\vdash(Basic\ f\#cs,css,Normal\ s) \rightarrow (cs,css,Normal\ (f\ s))$

| *Spec*: $(s,t) \in r \Longrightarrow \Gamma\vdash(Spec\ r\#cs,css,Normal\ s) \rightarrow (cs,css,Normal\ t)$
| *SpecStuck*: $\forall\, t.\ (s,t) \notin r \Longrightarrow \Gamma\vdash(Spec\ r\#cs,css,Normal\ s) \rightarrow (cs,css,Stuck)$

| *Seq*: $\Gamma\vdash(Seq\ c_1\ c_2\#cs,css,Normal\ s) \rightarrow (c_1\#c_2\#cs,css,Normal\ s)$

| *CondTrue*: $s\in b \Longrightarrow \Gamma\vdash(Cond\ b\ c_1\ c_2\#cs,css,Normal\ s) \rightarrow (c_1\#cs,css,Normal\ s)$
| *CondFalse*: $s\notin b \Longrightarrow \Gamma\vdash(Cond\ b\ c_1\ c_2\#cs,css,Normal\ s) \rightarrow (c_2\#cs,css,Normal\ s)$

| *WhileTrue*: $[\![s\in b]\!]$
$\Longrightarrow$
$\Gamma\vdash(While\ b\ c\#cs,css,Normal\ s) \rightarrow (c\#While\ b\ c\#cs,css,Normal\ s)$
| *WhileFalse*: $[\![s\notin b]\!]$
$\Longrightarrow$
$\Gamma\vdash(While\ b\ c\#cs,css,Normal\ s) \rightarrow (cs,css,Normal\ s)$

| *Call*: $\Gamma\ p=Some\ bdy \Longrightarrow$
$\Gamma\vdash(Call\ p\#cs,css,Normal\ s) \rightarrow ([bdy],(cs,Throw\#cs)\#css,Normal\ s)$

| *CallUndefined*: $\Gamma\ p=None \Longrightarrow$
$\Gamma\vdash(Call\ p\#cs,css,Normal\ s) \rightarrow (cs,css,Stuck)$

| *StuckProp*: $\Gamma\vdash(c\#cs,css,Stuck) \rightarrow (cs,css,Stuck)$
| *StuckPropBlock*: $\Gamma\vdash([],(nrms,abrs)\#css,Stuck) \rightarrow (nrms,css,Stuck)$

| *DynCom*: $\Gamma\vdash(DynCom\ c\#cs,css,Normal\ s) \rightarrow (c\ s\#cs,css,Normal\ s)$

| *Throw*: $\Gamma\vdash(Throw\#cs,css,Normal\ s) \rightarrow (cs,css,Abrupt\ s)$
| *Catch*: $\Gamma\vdash(Catch\ c_1\ c_2\#cs,css,Normal\ s) \rightarrow ([c_1],(cs,c_2\#cs)\#css,Normal\ s)$

lemmas *step-induct* = *step.induct* [*of* - (c,css,s) (c',css',s'), *split-format* (*complete*), *case-names*
Skip Guard GuardFault FaultProp FaultPropBlock AbruptProp ExitBlockNormal

ExitBlockAbrupt
Basic Spec SpecStuck Seq CondTrue CondFalse WhileTrue WhileFalse Call CallUndefined StuckProp StuckPropBlock DynCom Throw Catch, *induct set*]

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

inductive-cases *step-Normal-elim-cases* [*cases set*]:
$\Gamma\vdash(c\#cs,css,Fault\ f) \to u$
$\Gamma\vdash([],css,Fault\ f) \to u$
$\Gamma\vdash(c\#cs,css,Stuck) \to u$
$\Gamma\vdash([],css,Stuck) \to u$
$\Gamma\vdash([],(nrms,abrs)\#css,Normal\ s) \to u$
$\Gamma\vdash([],(nrms,abrs)\#css,Abrupt\ s) \to u$
$\Gamma\vdash(Skip\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Guard\ f\ g\ c\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Basic\ f\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Spec\ r\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Seq\ c1\ c2\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Cond\ b\ c1\ c2\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(While\ b\ c\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Call\ p\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(DynCom\ c\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Throw\#cs,css,Normal\ s) \to u$
$\Gamma\vdash(Catch\ c1\ c2\#cs,css,Normal\ s) \to u$

abbreviation
step-rtrancl :: $[('s,'p,'f)\ body,('s,'p,'f)\ config,('s,'p,'f)\ config] \Rightarrow bool$
$(\text{-}\vdash\ (\text{-}\ \to^*/\ \text{-})\ [81,81,81]\ 100)$
where
$\Gamma\vdash cs0 \to^* cs1 \quad == (step\ \Gamma)^{**}\ cs0\ cs1$

abbreviation

step-trancl :: $[('s,'p,'f)$ *body*$,('s,'p,'f)$ *config*$,('s,'p,'f)$ *config*$] \Rightarrow$ *bool*
$(\text{-}\vdash (\text{-} \rightarrow^{+}/ \text{-}) [81,81,81]\ 100)$
where
$\Gamma\vdash cs0 \rightarrow^{+} cs1 \quad == (step\ \Gamma)^{++}\ cs0\ cs1$

14.1.1 Structural Properties of Small Step Computations

lemma *Fault-app-steps*: $\Gamma\vdash(cs@xs,css,Fault\ f) \rightarrow^{*} (xs,css,Fault\ f)$
proof (*induct cs*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons c cs*)
  have $\Gamma\vdash(c\#cs@xs,\ css,\ Fault\ f) \rightarrow^{*} (xs,\ css,\ Fault\ f)$
  proof −
    have $\Gamma\vdash(c\#cs@xs,\ css,\ Fault\ f) \rightarrow (cs@xs,\ css,\ Fault\ f)$
      by (*rule step.FaultProp*)
    also
    have $\Gamma\vdash(cs@xs,\ css,\ Fault\ f) \rightarrow^{*} (xs,\ css,\ Fault\ f)$
      by (*rule Cons.hyps*)
    finally show *?thesis* **.**
  qed
  thus *?case*
    by *simp*
qed

lemma *Stuck-app-steps*: $\Gamma\vdash(cs@xs,css,Stuck) \rightarrow^{*} (xs,css,Stuck)$
proof (*induct cs*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons c cs*)
  have $\Gamma\vdash(c\#cs@xs,\ css,\ Stuck) \rightarrow^{*} (xs,\ css,\ Stuck)$
  proof −
    have $\Gamma\vdash(c\#cs@xs,\ css,\ Stuck) \rightarrow (cs@xs,\ css,\ Stuck)$
      by (*rule step.StuckProp*)
    also
    have $\Gamma\vdash(cs@xs,\ css,\ Stuck) \rightarrow^{*} (xs,\ css,\ Stuck)$
      by (*rule Cons.hyps*)
    finally show *?thesis* **.**
  qed
  thus *?case*
    by *simp*
qed

We can only append commands inside a block, if execution does not enter or exit a block.

lemma *app-step*:
  assumes *step*: $\Gamma\vdash(cs,css,s) \rightarrow (cs',css',t)$

shows $css=css' \Longrightarrow \Gamma\vdash(cs@xs,css,s) \rightarrow (cs'@xs,css',t)$
using *step*
apply *induct*
apply (*simp-all del*: *fun-upd-apply*,(*blast intro*: *step.intros*)+)
done

We can append whole blocks, without interfering with the actual block. Outer blocks do not influence execution of inner blocks.

lemma *app-css-step*:
assumes *step*: $\Gamma\vdash(cs,css,s) \rightarrow (cs',css',t)$
shows $\Gamma\vdash(cs,css@xs,s) \rightarrow (cs',css'@xs,t)$
using *step*
apply *induct*
apply (*simp-all del*: *fun-upd-apply*,(*blast intro*: *step.intros*)+)
done

ML ‹
ML-Thms.bind-thm (*trancl-induct3*, *Split-Rule.split-rule* @{*context*}
(*Rule-Insts.read-instantiate* @{*context*}
[(((*a*, *0*), *Position.none*), (*ax*, *ay*, *az*)),
(((*b*, *0*), *Position.none*), (*bx*, *by*, *bz*))] []
@{*thm tranclp-induct*}));
›

lemma *app-css-steps*:
assumes *step*: $\Gamma\vdash(cs,css,s) \rightarrow^+ (cs',css',t)$
shows $\Gamma\vdash(cs,css@xs,s) \rightarrow^+ (cs',css'@xs,t)$
apply(*rule trancl-induct3* [*OF step*])
apply (*rule app-css-step* [*THEN tranclp.r-into-trancl* [*of step* Γ]],*assumption*)
apply(*blast intro*:*app-css-step tranclp-trans*)
done

lemma *step-Cons'*:
assumes *step*: $\Gamma\vdash(ccs,css,s) \rightarrow (cs',css',t)$
shows
$\bigwedge c\ cs.\ ccs=c\#cs \Longrightarrow \exists\, css''.\ css'=css''@css\ \wedge$
$(if\ css''=[]\ then\ \exists\, p.\ cs'=p@cs$
$else\ (\exists\, pnorm\ pabr.\ css''=[(pnorm@cs,pabr@cs)]))$
using *step*
by *induct force*+

lemma *step-Cons*:
assumes *step*: $\Gamma\vdash(c\#cs,css,s) \rightarrow (cs',css',t)$
shows $\exists\, pcss.\ css'=pcss@css\ \wedge$
$(if\ pcss=[]\ then\ \exists\, ps.\ cs'=ps@cs$
$else\ (\exists\, pcs\text{-}normal\ pcs\text{-}abrupt.\ pcss=[(pcs\text{-}normal@cs,pcs\text{-}abrupt@cs)]))$
using *step-Cons'* [*OF step*]
by *blast*

lemma *step-Nil'*:
  assumes *step*: $\Gamma\vdash(cs,asscss,s) \rightarrow (cs',css',t)$
  shows
  $\bigwedge ass.\ [\![cs=[];\ asscss=ass@css;\ ass\neq Nil]\!] \Longrightarrow$
    $css'=tl\ ass@css \wedge$
    $(case\ s\ of$
      $Abrupt\ s' \Rightarrow cs'=snd\ (hd\ ass) \wedge t=Normal\ s'$
    $|\ \_ \Rightarrow cs'=fst\ (hd\ ass) \wedge t=s)$
using *step*
by (*induct*) (*fastforce simp add*: *neq-Nil-conv*)+

lemma *step-Nil*:
  assumes *step*: $\Gamma\vdash([],ass@css,s) \rightarrow (cs',css',t)$
  assumes *ass-not-Nil*: $ass\neq[]$
  shows $css'=tl\ ass@css \wedge$
    $(case\ s\ of$
      $Abrupt\ s' \Rightarrow cs'=snd\ (hd\ ass) \wedge t=Normal\ s'$
    $|\ \_ \Rightarrow cs'=fst\ (hd\ ass) \wedge t=s)$
  using *step-Nil'* [*OF step - - ass-not-Nil*]
  by *simp*

lemma *step-Nil''*:
  assumes *step*: $\Gamma\vdash([],(pcs\text{-}normal,pcs\text{-}abrupt)\#pcss@css,s) \rightarrow (cs',pcss@css,t)$
  shows $(case\ s\ of$
      $Abrupt\ s' \Rightarrow cs'=pcs\text{-}abrupt \wedge t=Normal\ s'$
    $|\ \_ \Rightarrow cs'=pcs\text{-}normal \wedge t=s)$
  using *step-Nil'* [*OF step*, **where** $ass = (pcs\text{-}normal,pcs\text{-}abrupt)\#pcss$ **and** $css=css$]
  by (*auto split*: *xstate.splits*)

lemma *drop-suffix-css-step'*:
assumes *step*: $\Gamma\vdash(cs,cssxs,s) \rightarrow (cs',css'xs,t)$
shows $\bigwedge css\ css'\ xs.\ [\![cssxs = css@xs;\ css'xs=css'@xs]\!]$
  $\Longrightarrow \Gamma\vdash(cs,css,s) \rightarrow (cs',css',t)$
using *step*
apply *induct*
apply (*fastforce intro*: *step.intros*)+
done

lemma *drop-suffix-css-step*:
assumes *step*: $\Gamma\vdash(cs,pcss@css,s) \rightarrow (cs',pcss'@css,t)$
shows $\Gamma\vdash(cs,pcss,s) \rightarrow (cs',pcss',t)$
using *step* **by** (*blast intro*: *drop-suffix-css-step'*)

lemma *drop-suffix-hd-css-step'*:
  assumes *step*: $\Gamma\vdash\ (pcs,css,s) \rightarrow (cs',css'css,t)$
  shows $\bigwedge p\ ps\ cs\ pnorm\ pabr.\ [\![pcs=p\#ps@cs;\ css'css=(pnorm@cs,pabr@cs)\#css]\!]$
    $\Longrightarrow \Gamma\vdash\ (p\#ps,css,s) \rightarrow (cs',(pnorm,pabr)\#css,t)$
using *step*

by *induct* (*force intro*: *step.intros*)+

lemma *drop-suffix-hd-css-step''*:
  assumes *step*: $\Gamma\vdash$ ($p\#ps@cs$,css,s) $\rightarrow$ (cs',($pnorm@cs$,$pabr@cs$)$\#css$,t)
  shows $\Gamma\vdash$ ($p\#ps$,css,s) $\rightarrow$ (cs',($pnorm$,$pabr$)$\#css$,t)
using *drop-suffix-hd-css-step'* [*OF step*]
by *auto*

lemma *drop-suffix-hd-css-step*:
  assumes *step*: $\Gamma\vdash$ ($p\#ps@cs$,css,s) $\rightarrow$ (cs',[($pnorm@ps@cs$,$pabr@ps@cs$)]$@css$,t)
  shows $\Gamma\vdash$ ($p\#ps$,css,s) $\rightarrow$ (cs',[($pnorm@ps$,$pabr@ps$)]$@css$,t)
proof −
  from *step drop-suffix-hd-css-step''* [*of - p ps cs css s cs' pnorm@ps pabr@ps t*]
  show *?thesis*
    by *auto*
qed

lemma *drop-suffix'*:
  assumes *step*: $\Gamma\vdash$($csxs$,css,s) $\rightarrow$ ($cs'xs$,css',t)
  shows $\bigwedge xs\ cs\ cs'$. $[\![css=css'; csxs=cs@xs; cs'xs = cs'@xs; cs\neq[]\ ]\!]$
        $\Longrightarrow \Gamma\vdash$($cs$,$css$,$s$) $\rightarrow$ (cs',css,t)
using *step*
apply *induct*
apply (*fastforce intro*: *step.intros simp add*: *neq-Nil-conv*)+
done

lemma *drop-suffix*:
  assumes *step*: $\Gamma\vdash$($c\#cs@xs$,css,s) $\rightarrow$ ($cs'@xs$,css,t)
  shows $\Gamma\vdash$($c\#cs$,css,s) $\rightarrow$ (cs',css,t)
  by(*rule drop-suffix'* [*OF step - - -*]) *auto*

lemma *drop-suffix-same-css-step*:
  assumes *step*: $\Gamma\vdash$($cs@xs$,css,s) $\rightarrow$ ($cs'@xs$,css,t)
  assumes *not-Nil*: $cs\neq[]$
  shows $\Gamma\vdash$(cs,xss,s) $\rightarrow$ (cs',xss,t)
proof−
  from *drop-suffix'* [*OF step - - - not-Nil*]
  have $\Gamma\vdash$(cs,css,s) $\rightarrow$ (cs',css,t)
    by *auto*
  with *drop-suffix-css-step* [*of - cs* [] *css s cs'* [] *t*]
  have $\Gamma\vdash$ (cs, [], s) $\rightarrow$ (cs', [], t)
    by *auto*
  from *app-css-step* [*OF this*]
  show *?thesis*
    by *auto*
qed

lemma *Cons-change-css-step*:
  assumes *step*: $\Gamma\vdash$ (cs,css,s) $\rightarrow$ (cs',$css'@css$,t)

shows Γ⊢ (*cs*,*xss*,*s*) → (*cs'*,*css'*@*xss*,*t*)
proof −
from *step*
drop-suffix-css-step [**where** *cs*=*cs* **and** *pcss*=[] **and** *css*=*css* **and** *s*=*s*
and *cs'*=*cs'* **and** *pcss'*=*css'* **and** *t*=*t*]
have Γ⊢ (*cs*, [], *s*) → (*cs'*, *css'*, *t*)
by *auto*
from *app-css-step* [**where** *xs*=*xss*, *OF this*]
show *?thesis*
by *auto*
qed

lemma *Nil-change-css-step*:
assumes *step*: Γ⊢([],*ass*@*css*,*s*) → (*cs'*,*ass'*@*css*,*t*)
assumes *ass-not-Nil*: *ass*≠[]
shows Γ⊢([],*ass*@*xss*,*s*) → (*cs'*,*ass'*@*xss*,*t*)
proof −
from *step drop-suffix-css-step* [*of* - [] *ass css s cs' ass' t*]
have Γ⊢ ([], *ass*, *s*) → (*cs'*, *ass'*, *t*)
by *auto*
from *app-css-step* [**where** *xs*=*xss*, *OF this*]
show *?thesis*
by *auto*
qed

14.1.2 Equivalence between Big and Small-Step Semantics

lemma *exec-impl-steps*:
assumes *exec*: Γ⊢⟨*c*,*s*⟩ ⇒ *t*
shows ⋀*cs css*. Γ⊢(*c*#*cs*,*css*,*s*) →* (*cs*,*css*,*t*)
using *exec*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*blast intro*: *step.Skip*)
next
case *Guard* **thus** *?case* **by** (*blast intro*: *step.Guard rtranclp-trans*)
next
case *GuardFault* **thus** *?case* **by** (*blast intro*: *step.GuardFault*)
next
case *FaultProp* **thus** *?case* **by** (*blast intro*: *step.FaultProp*)
next
case *Basic* **thus** *?case* **by** (*blast intro*: *step.Basic*)
next
case *Spec* **thus** *?case* **by** (*blast intro*: *step.Spec*)
next
case *SpecStuck* **thus** *?case* **by** (*blast intro*: *step.SpecStuck*)
next
case *Seq* **thus** *?case* **by** (*blast intro*: *step.Seq rtranclp-trans*)
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* **thus** *?case* **by** (*blast intro*: *step.WhileTrue rtranclp-trans*)
next
  case *WhileFalse* **thus** *?case* **by** (*blast intro*: *step.WhileFalse*)
next
  case (*Call p bdy s s' cs css*)
  have *bdy*: Γ *p* = *Some bdy* **by** *fact*
  have *steps-body*: $\Gamma\vdash([bdy],(cs,Throw\#cs)\#css,Normal\ s) \rightarrow^*$
                    $([],(cs,Throw\#cs)\#css,\ s')$ **by** *fact*
  show *?case*
  proof (*cases s'*)
    case (*Normal s''*)
    note *steps-body*
    also from *Normal* **have** $\Gamma\vdash([],(cs,Throw\#cs)\#css,\ s') \rightarrow (cs,css,s')$
      by (*auto intro*: *step.intros*)
    finally show *?thesis*
      using *bdy*
      by (*blast intro*: *step.Call rtranclp-trans*)
  next
    case (*Abrupt s''*)
    with *steps-body*
    have $\Gamma\vdash([bdy],(cs,Throw\#cs)\#css,Normal\ s) \rightarrow^*$
           $([],(cs,Throw\#cs)\#css,\ Abrupt\ s'')$ **by** *simp*
     also have $\Gamma\vdash([],(cs,Throw\#cs)\#css,\ Abrupt\ s'') \rightarrow (Throw\#cs,css,Normal$
$s'')$
      by (*rule ExitBlockAbrupt*)
    also have $\Gamma\vdash(Throw\#cs,css,Normal\ s'') \rightarrow (cs,css,Abrupt\ s'')$
      by (*rule Throw*)
    finally show *?thesis*
      using *bdy Abrupt*
      by (*auto intro*: *step.Call rtranclp-trans*)
  next
    case *Fault*
    note *steps-body*
    also from *Fault* **have** $\Gamma\vdash([],(cs,Throw\#cs)\#css,\ s') \rightarrow (cs,css,s')$
      by (*auto intro*: *step.intros*)
    finally show *?thesis*
      using *bdy*
      by (*blast intro*: *step.Call rtranclp-trans*)
  next
    case *Stuck*
    note *steps-body*
    also from *Stuck* **have** $\Gamma\vdash([],(cs,Throw\#cs)\#css,\ s') \rightarrow (cs,css,s')$
      by (*auto intro*: *step.intros*)
    finally show *?thesis*
      using *bdy*
      by (*blast intro*: *step.Call rtranclp-trans*)

qed
next
case (*CallUndefined p s cs css*)
have *undef*: Γ *p* = *None* **by** *fact*
hence $\Gamma\vdash$(*Call p* # *cs*, *css*, *Normal s*) $\rightarrow$ (*cs*, *css*, *Stuck*)
by (*rule step.CallUndefined*)
thus *?case* **..**
next
case *StuckProp* **thus** *?case* **by** (*blast intro*: *step.StuckProp rtrancl-trans*)
next
case *DynCom* **thus** *?case* **by** (*blast intro*: *step.DynCom rtranclp-trans*)
next
case *Throw* **thus** *?case* **by** (*blast intro*: *step.Throw*)
next
case *AbruptProp* **thus** *?case* **by** (*blast intro*: *step.AbruptProp*)
next
case (*CatchMatch* c_1 *s s'* c_2 *s'' cs css*)
have *steps-c1*: $\Gamma\vdash$([c_1],(*cs*,c_2#*cs*)#*css*,*Normal s*) $\rightarrow^*$
([],(*cs*,c_2#*cs*)#*css*,*Abrupt s'*) **by** *fact*
also
have $\Gamma\vdash$([],(*cs*,c_2#*cs*)#*css*,*Abrupt s'*) $\rightarrow$ (c_2#*cs*,*css*,*Normal s'*)
by (*rule ExitBlockAbrupt*)
also
have *steps-c2*: $\Gamma\vdash$(c_2#*cs*,*css*,*Normal s'*) $\rightarrow^*$ (*cs*,*css*,*s''*) **by** *fact*
finally
show $\Gamma\vdash$(*Catch* c_1 c_2 # *cs*, *css*, *Normal s*) $\rightarrow^*$ (*cs*, *css*, *s''*)
by (*blast intro*: *step.Catch rtranclp-trans*)
next
case (*CatchMiss* c_1 *s s'* c_2 *cs css*)
assume *notAbr*: $\neg$ *isAbr s'*
have *steps-c1*: $\Gamma\vdash$([c_1],(*cs*,c_2#*cs*)#*css*,*Normal s*) $\rightarrow^*$ ([],(*cs*,c_2#*cs*)#*css*,*s'*) **by** *fact*
show $\Gamma\vdash$(*Catch* c_1 c_2 # *cs*, *css*, *Normal s*) $\rightarrow^*$ (*cs*, *css*, *s'*)
proof (*cases s'*)
case (*Normal w*)
with *steps-c1*
have $\Gamma\vdash$([c_1],(*cs*,c_2#*cs*)#*css*,*Normal s*) $\rightarrow^*$ ([],(*cs*,c_2#*cs*)#*css*,*Normal w*)
by *simp*
also
have $\Gamma\vdash$([],(*cs*,c_2#*cs*)#*css*,*Normal w*) $\rightarrow$ (*cs*,*css*,*Normal w*)
by (*rule ExitBlockNormal*)
finally show *?thesis* **using** *Normal*
by (*auto intro*: *step.Catch rtranclp-trans*)
next
case *Abrupt* **with** *notAbr* **show** *?thesis* **by** *simp*
next
case (*Fault f*)
with *steps-c1*
have $\Gamma\vdash$([c_1],(*cs*,c_2#*cs*)#*css*,*Normal s*) $\rightarrow^*$ ([],(*cs*,c_2#*cs*)#*css*,*Fault f*)

by *simp*
also
have $\Gamma\vdash([],(cs,c_2\#cs)\#css,Fault\ f) \rightarrow (cs,css,Fault\ f)$
by (*rule FaultPropBlock*)
finally show *?thesis* **using** *Fault*
by (*auto intro*: *step.Catch rtranclp-trans*)
next
case *Stuck*
with *steps-c1*
have $\Gamma\vdash([c_1],(cs,c_2\#cs)\#css,Normal\ s) \rightarrow^* ([],(cs,c_2\#cs)\#css,Stuck)$
by *simp*
also
have $\Gamma\vdash([],(cs,c_2\#cs)\#css,Stuck) \rightarrow (cs,css,Stuck)$
by (*rule StuckPropBlock*)
finally show *?thesis* **using** *Stuck*
by (*auto intro*: *step.Catch rtranclp-trans*)
qed
qed

inductive *execs*::$[('s,'p,'f)\ body,('s,'p,'f)\ com\ list,$
$('s,'p,'f)\ continuation\ list,$
$('s,'f)\ xstate,('s,'f)\ xstate] \Rightarrow bool$
$(\text{-}\vdash \langle\text{-},\text{-},\text{-}\rangle \Rightarrow \text{-}\ [50,50,50,50,50]\ 50)$
for Γ:: $('s,'p,'f)\ body$
where
Nil: $\Gamma\vdash\langle[],[],s\rangle \Rightarrow s$

| *ExitBlockNormal*: $\Gamma\vdash\langle nrms,css,Normal\ s\rangle \Rightarrow t$
$\Longrightarrow$
$\Gamma\vdash\langle[],(nrms,abrs)\#css,Normal\ s\rangle \Rightarrow t$

| *ExitBlockAbrupt*: $\Gamma\vdash\langle abrs,css,Normal\ s\rangle \Rightarrow t$
$\Longrightarrow$
$\Gamma\vdash\langle[],(nrms,abrs)\#css,Abrupt\ s\rangle \Rightarrow t$

| *ExitBlockFault*: $\Gamma\vdash\langle nrms,css,Fault\ f\rangle \Rightarrow t$
$\Longrightarrow$
$\Gamma\vdash\langle[],(nrms,abrs)\#css,Fault\ f\rangle \Rightarrow t$

| *ExitBlockStuck*: $\Gamma\vdash\langle nrms,css,Stuck\rangle \Rightarrow t$
$\Longrightarrow$
$\Gamma\vdash\langle[],(nrms,abrs)\#css,Stuck\rangle \Rightarrow t$

| *Cons*: $[\![\Gamma\vdash\langle c,s\rangle \Rightarrow t;\ \Gamma\vdash\langle cs,css,t\rangle \Rightarrow u]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle c\#cs,css,s\rangle \Rightarrow u$

inductive-cases *execs-elim-cases* [*cases set*]:
$\Gamma\vdash\langle[],css,s\rangle \Rightarrow t$
$\Gamma\vdash\langle c\#cs,css,s\rangle \Rightarrow t$

ML ‹
ML-Thms.bind-thm (*converse-rtrancl-induct3*, *Split-Rule.split-rule* @{*context*}
(*Rule-Insts.read-instantiate* @{*context*}
[(((*a*, *0*), *Position.none*), (*cs*, *css*, *s*)),
(((*b*, *0*), *Position.none*), (*cs'*, *css'*, *t*))] []
@{*thm converse-rtranclp-induct*}));
›

lemma *execs-Fault-end*:
assumes *execs*: $\Gamma\vdash\langle cs,css,s\rangle \Rightarrow t$ **shows** $s=Fault\ f \Longrightarrow t=Fault\ f$
using *execs*
by (*induct*) (*auto dest*: *Fault-end*)

lemma *execs-Stuck-end*:
assumes *execs*: $\Gamma\vdash\langle cs,css,s\rangle \Rightarrow t$ **shows** $s=Stuck \Longrightarrow t=Stuck$
using *execs*
by (*induct*) (*auto dest*: *Stuck-end*)

theorem *steps-impl-execs*:
assumes *steps*: $\Gamma\vdash(cs,css,s) \rightarrow^* ([],[],t)$
shows $\Gamma\vdash\langle cs,css,s\rangle \Rightarrow t$
using *steps*
proof (*induct rule*: *converse-rtrancl-induct3* [*consumes 1*])
show $\Gamma\vdash\langle[],[],t\rangle \Rightarrow t$ **by** (*rule execs.Nil*)
next
fix *cs css s cs' css' w*
assume *step*: $\Gamma\vdash(cs,css, s) \rightarrow (cs',css', w)$
assume *execs*: $\Gamma\vdash\langle cs',css',w\rangle \Rightarrow t$
from *step*
show $\Gamma\vdash\langle cs,css,s\rangle \Rightarrow t$
proof (*cases*)
case (*Catch c1 c2 cs s*)
with *execs* **obtain** *t'* **where**
exec-c1: $\Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow t'$ **and**
execs-rest: $\Gamma\vdash\langle[],(cs,\ c2\ \#\ cs)\ \#\ css,t'\rangle \Rightarrow t$
by (*clarsimp elim*!: *execs-elim-cases*)
have $\Gamma\vdash\langle Catch\ c1\ c2\ \#\ cs,css,Normal\ s\rangle \Rightarrow t$
proof (*cases t'*)
case (*Normal t''*)
with *exec-c1* **have** $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow t'$
by (*auto intro*: *exec.CatchMiss*)
moreover
from *execs-rest Normal* **have** $\Gamma\vdash\langle cs,css,t'\rangle \Rightarrow t$
by (*cases*) *auto*

ultimately show *?thesis*
by (*rule execs.Cons*)
next
case (*Abrupt* t'')
from *execs-rest Abrupt* **have** $\Gamma\vdash\langle c2\#cs,css,Normal\ t''\rangle \Rightarrow t$
by (*cases*) *auto*
then obtain v **where**
exec-c2: $\Gamma\vdash\langle c2,Normal\ t''\rangle \Rightarrow v$ **and**
rest: $\Gamma\vdash\langle cs,css,v\rangle \Rightarrow t$
by *cases*
from *exec-c1 Abrupt exec-c2*
have $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow v$
by $-$ (*rule exec.CatchMatch*, *auto*)
from *this rest*
show *?thesis*
by (*rule execs.Cons*)
next
case (*Fault f*)
with *exec-c1* **have** $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow Fault\ f$
by (*auto intro*: *exec.intros*)
moreover from *execs-rest Fault* **have** $\Gamma\vdash\langle cs,css,Fault\ f\rangle \Rightarrow t$
by (*cases*) *auto*
ultimately show *?thesis*
by (*rule execs.Cons*)
next
case *Stuck*
with *exec-c1* **have** $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow Stuck$
by (*auto intro*: *exec.intros*)
moreover from *execs-rest Stuck* **have** $\Gamma\vdash\langle cs,css,Stuck\rangle \Rightarrow t$
by (*cases*) *auto*
ultimately show *?thesis*
by (*rule execs.Cons*)
qed
with *Catch* **show** *?thesis* **by** *simp*
next
case (*Call p bdy cs s*)
have *bdy*: $\Gamma\ p = Some\ bdy$ **by** *fact*
from *Call execs* **obtain** t' **where**
exec-body: $\Gamma\vdash\langle bdy,Normal\ s\rangle \Rightarrow t'$ **and**
execs-rest:
$\Gamma\vdash\langle[],(cs,Throw\#cs)\#css\ ,t'\rangle \Rightarrow t$
by (*clarsimp elim*!: *execs-elim-cases*)
have $\Gamma\vdash\langle Call\ p\ \#\ cs,css,Normal\ s\rangle \Rightarrow t$
proof (*cases* t')
case (*Normal* t'')
with *exec-body bdy*
have $\Gamma\vdash\langle Call\ p\ ,Normal\ s\rangle \Rightarrow Normal\ t''$
by (*auto intro*: *exec.intros*)
moreover

      from *execs-rest Normal*
      have $\Gamma\vdash\langle cs,css\ ,Normal\ t''\rangle \Rightarrow t$
        by *cases auto*
      ultimately show *?thesis* **by** (*rule execs.Cons*)
    next
      case (*Abrupt t''*)
      with *exec-body bdy*
      have $\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow Abrupt\ t''$
        by (*auto intro*: *exec.intros*)
      moreover
      from *execs-rest Abrupt* **have**
        $\Gamma\vdash\langle Throw\ \#\ cs,css,Normal\ t''\rangle \Rightarrow t$
        by (*cases*) *auto*
      then obtain *v* **where** *v*: $\Gamma\vdash\langle Throw,Normal\ t''\rangle \Rightarrow v$ $\Gamma\vdash\langle cs,css,v\rangle \Rightarrow t$
        by (*clarsimp elim*!: *execs-elim-cases*)
      moreover from *v* **have** $v=Abrupt\ t''$
        by (*auto elim*: *exec-Normal-elim-cases*)
      ultimately
      show *?thesis* **by** (*auto intro*: *execs.Cons*)
    next
      case (*Fault f*)
      with *exec-body bdy* **have** $\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow Fault\ f$
        by (*auto intro*: *exec.intros*)
      moreover from *execs-rest Fault* **have** $\Gamma\vdash\langle cs,css,Fault\ f\rangle \Rightarrow t$
        by (*cases*) (*auto elim*: *execs-elim-cases dest*: *Fault-end*)
      ultimately
      show *?thesis* **by** (*rule execs.Cons*)
    next
      case *Stuck*
      with *exec-body bdy* **have** $\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow Stuck$
        by (*auto intro*: *exec.intros*)
      moreover from *execs-rest Stuck* **have** $\Gamma\vdash\langle cs,css,Stuck\rangle \Rightarrow t$
        by (*cases*) (*auto elim*: *execs-elim-cases dest*: *Stuck-end*)
      ultimately
      show *?thesis* **by** (*rule execs.Cons*)
    qed
    with *Call* **show** *?thesis* **by** *simp*
  qed (*insert execs*,
      (*blast intro*:*execs.intros exec.intros elim*!: *execs-elim-cases*)+)
qed

theorem *steps-impl-exec*:
  assumes *steps*: $\Gamma\vdash([c],[],s) \rightarrow^* ([],[],t)$
  shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
using *steps-impl-execs* [*OF steps*]
by (*blast elim*: *execs-elim-cases*)

corollary *steps-eq-exec*: $\Gamma\vdash([c],[],s) \rightarrow^* ([],[],t) = \Gamma\vdash\langle c,s\rangle \Rightarrow t$
  by (*blast intro*: *steps-impl-exec exec-impl-steps*)

14.2 Infinite Computations: *inf* Γ *cs css s*

definition *inf* ::
$[('s,'p,'f)\ body,('s,'p,'f)\ com\ list,('s,'p,'f)\ continuation\ list,('s,'f)\ xstate]$
$\Rightarrow bool$
where $inf\ \Gamma\ cs\ css\ s = (\exists f.\ f\ 0 = (cs,css,s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f(Suc\ i)))$

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

14.3 Equivalence of Termination and Absence of Infinite Computations

inductive *terminatess*:: $[('s,'p,'f)\ body,('s,'p,'f)\ com\ list,$
$('s,'p,'f)\ continuation\ list,('s,'f)\ xstate] \Rightarrow bool$
$(\text{-}\vdash\text{-},\text{-} \Downarrow \text{-}\ [60,20,60]\ 89)$
for Γ::$('s,'p,'f)\ body$
where
Nil: $\Gamma\vdash[\,],[\,]\Downarrow s$

| *ExitBlockNormal*: $\Gamma\vdash nrms,css\Downarrow Normal\ s$
$\Longrightarrow$
$\Gamma\vdash[\,],(nrms,abrs)\#css\Downarrow Normal\ s$

| *ExitBlockAbrupt*: $\Gamma\vdash abrs,css\Downarrow Normal\ s$
$\Longrightarrow$
$\Gamma\vdash[\,],(nrms,abrs)\#css\Downarrow Abrupt\ s$

| *ExitBlockFault*: $\Gamma\vdash nrms,css\Downarrow Fault\ f$
$\Longrightarrow$
$\Gamma\vdash[\,],(nrms,abrs)\#css\Downarrow Fault\ f$

| *ExitBlockStuck*: $\Gamma\vdash nrms,css\Downarrow Stuck$
$\Longrightarrow$
$\Gamma\vdash[\,],(nrms,abrs)\#css\Downarrow Stuck$

| *Cons*: $[\![\Gamma\vdash c\downarrow s;\ (\forall t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \longrightarrow \Gamma\vdash cs,css\Downarrow t)]\!]$
$\Longrightarrow$
$\Gamma\vdash c\#cs,css\Downarrow s$

inductive-cases *terminatess-elim-cases* [*cases set*]:
$\Gamma\vdash[\,],css\Downarrow t$
$\Gamma\vdash c\#cs,css\Downarrow t$

lemma *terminatess-Fault*: $\bigwedge cs.\ \Gamma\vdash cs,css\Downarrow Fault\ f$
proof (*induct css*)
case *Nil*
show $\Gamma\vdash cs,[\,]\Downarrow Fault\ f$

```
  proof (induct cs)
    case Nil show Γ⊢[],[]⇓Fault f by (rule terminatess.Nil)
  next
    case (Cons c cs)
    thus ?case
      by (auto intro: terminatess.intros terminates.intros dest: Fault-end)
  qed
next
  case (Cons d css)
  have hyp: ⋀cs. Γ⊢cs,css⇓Fault f by fact
  obtain nrms abrs where d: d=(nrms,abrs)  by (cases d) auto
  have Γ⊢cs,(nrms,abrs)#css⇓Fault f
  proof (induct cs)
    case Nil
    show Γ⊢[],(nrms, abrs) # css⇓Fault f
      by (rule terminatess.ExitBlockFault) (rule hyp)
  next
    case (Cons c cs)
    have hyp1: Γ⊢cs,(nrms, abrs) # css⇓Fault f by fact
    show Γ⊢c#cs,(nrms, abrs)#css⇓Fault f
      by (auto intro: hyp1 terminatess.Cons terminates.intros dest: Fault-end)
  qed
  with d show ?case by simp
qed

lemma terminatess-Stuck: ⋀cs. Γ⊢cs,css⇓Stuck
proof (induct css)
  case Nil
  show Γ⊢cs,[]⇓Stuck
  proof (induct cs)
    case Nil show Γ⊢[],[]⇓Stuck by (rule terminatess.Nil)
  next
    case (Cons c cs)
    thus ?case
      by (auto intro: terminatess.intros terminates.intros dest: Stuck-end)
  qed
next
  case (Cons d css)
  have hyp: ⋀cs. Γ⊢cs,css⇓Stuck by fact
  obtain nrms abrs where d: d=(nrms,abrs) by (cases d) auto
  have Γ⊢cs,(nrms,abrs)#css⇓Stuck
  proof (induct cs)
    case Nil
    show Γ⊢[],(nrms, abrs) # css⇓Stuck
      by (rule terminatess.ExitBlockStuck) (rule hyp)
  next
    case (Cons c cs)
    have hyp1: Γ⊢cs,(nrms, abrs) # css⇓Stuck by fact
    show Γ⊢c#cs,(nrms, abrs)#css⇓Stuck
```

by (*auto intro*: *hyp1 terminatess.Cons terminates.intros dest*: *Stuck-end*)
qed
with *d* **show** *?case* **by** *simp*
qed

lemma *Basic-terminates*: $\Gamma\vdash Basic\ f \downarrow t$
by (*cases t*) (*auto intro*: *terminates.intros*)

lemma *step-preserves-terminations*:
assumes *step*: $\Gamma\vdash(cs,css,s) \rightarrow (cs',css',t)$
shows $\Gamma\vdash cs,css \Downarrow s \Longrightarrow \Gamma\vdash cs',css' \Downarrow t$
using *step*
proof (*induct*)
case *Skip* **thus** *?case*
by (*auto elim*: *terminates-Normal-elim-cases terminatess-elim-cases*
intro: *exec.intros*)
next
case *Guard* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
intro: *terminatess.intros terminates.intros exec.intros*)
next
case *GuardFault* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
intro: *terminatess.intros terminates.intros exec.intros*)
next
case *FaultProp* **show** *?case* **by** (*rule terminatess-Fault*)
next
case *FaultPropBlock* **show** *?case* **by** (*rule terminatess-Fault*)
next
case *AbruptProp* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases*
intro: *terminatess.intros*)
next
case *ExitBlockNormal* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases*
intro: *terminatess.intros*)
next
case *ExitBlockAbrupt* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases*
intro: *terminatess.intros*)
next
case *Basic* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
intro: *terminatess.intros terminates.intros exec.intros*)
next
case *Spec* **thus** *?case*
by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
intro: *terminatess.intros terminates.intros exec.intros*)

next
  case *SpecStuck* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case *Seq* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case *CondTrue* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case *CondFalse* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case *WhileTrue* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case *WhileFalse* **thus** *?case*
    by (*blast elim*: *terminatess-elim-cases terminates-Normal-elim-cases*
          intro: *terminatess.intros terminates.intros exec.intros*)
next
  case (*Call p bdy cs css s*)
  have *bdy*: Γ *p* = *Some bdy* **by** *fact*
  from *Call* **obtain**
    term-body: $\Gamma\vdash$*bdy* $\downarrow$ *Normal s* **and**
    term-rest: $\forall$ *t*. $\Gamma\vdash\langle$*Call p*,*Normal s*$\rangle \Rightarrow$ *t* $\longrightarrow$ $\Gamma\vdash$*cs*,*css*$\Downarrow$*t*
    by (*fastforce elim*!: *terminatess-elim-cases terminates-Normal-elim-cases*)
  show $\Gamma\vdash$[*bdy*],(*cs*,*Throw* # *cs*)#*css*$\Downarrow$*Normal s*
  proof (*rule terminatess.Cons* [*OF term-body*],*clarsimp*)
    fix *t*
    assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal s*$\rangle \Rightarrow$ *t*
    show $\Gamma\vdash$[],(*cs*,*Throw* # *cs*) # *css*$\Downarrow$*t*
    proof (*cases t*)
      case (*Normal t'*)
      with *exec-body bdy*
      have $\Gamma\vdash\langle$*Call p*,*Normal s*$\rangle \Rightarrow$ *Normal t'*
        by (*auto intro*: *exec.intros*)
      with *term-rest* **have** $\Gamma\vdash$*cs*,*css*$\Downarrow$*Normal t'*
        by *iprover*
      with *Normal* **show** *?thesis*
        by (*auto intro*: *terminatess.intros terminates.intros*
              elim: *exec-Normal-elim-cases*)
    next
      case (*Abrupt t'*)
      with *exec-body bdy*

```
      have Γ⊢⟨Call p,Normal s⟩ ⇒ Abrupt t'
        by (auto intro: exec.intros)
      with term-rest have Γ⊢cs,css⇓Abrupt t'
        by iprover
      with Abrupt show ?thesis
        by (fastforce intro: terminatess.intros terminates.intros
                   elim: exec-Normal-elim-cases)
    next
      case Fault
      thus ?thesis
        by (iprover intro: terminatess-Fault)
    next
      case Stuck
      thus ?thesis
        by (iprover intro: terminatess-Stuck)
    qed
  qed
next
  case CallUndefined thus ?case
    by (iprover intro: terminatess-Stuck)
next
  case StuckProp show ?case by (rule terminatess-Stuck)
next
  case StuckPropBlock show ?case by (rule terminatess-Stuck)
next
  case DynCom thus ?case
    by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
            intro: terminatess.intros terminates.intros exec.intros)
next
  case Throw thus ?case
    by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
            intro: terminatess.intros terminates.intros exec.intros)
next
  case (Catch c1 c2 cs css s)
  then obtain
    term-c1: Γ⊢c1 ↓ Normal s and
    term-c2: ∀ s'. Γ⊢⟨c1,Normal s⟩ ⇒ Abrupt s' ⟶ Γ⊢c2 ↓ Normal s'and
    term-rest: ∀ t. Γ⊢⟨Catch c1 c2,Normal s⟩ ⇒ t ⟶ Γ⊢cs,css⇓t
    by (clarsimp elim!: terminatess-elim-cases terminates-Normal-elim-cases)
  show Γ⊢[c1],(cs, c2 # cs) # css⇓Normal s
  proof (rule terminatess.Cons [OF term-c1],clarsimp)
    fix t
    assume exec-c1: Γ⊢⟨c1,Normal s⟩ ⇒ t
    show Γ⊢[],(cs, c2 # cs) # css⇓t
    proof (cases t)
      case (Normal t')
      with exec-c1 have Γ⊢⟨Catch c1 c2,Normal s⟩ ⇒ t
        by (auto intro: exec.intros)
      with term-rest have Γ⊢cs,css⇓t
```

```
      by iprover
    with Normal show ?thesis
      by (iprover intro: terminatess.intros)
  next
    case (Abrupt t')
    with exec-c1 term-c2 have Γ⊢c2 ↓ Normal t'
      by auto
    moreover
    {
      fix w
      assume exec-c2: Γ⊢⟨c2,Normal t'⟩ ⇒ w
      have Γ⊢cs,css⇓w
      proof -
        from exec-c1 Abrupt exec-c2
        have Γ⊢⟨Catch c1 c2,Normal s⟩ ⇒ w
          by (auto intro: exec.intros)
        with term-rest show ?thesis by simp
      qed
    }
    ultimately
    show ?thesis using Abrupt
      by (auto intro: terminatess.intros)
  next
    case Fault thus ?thesis
      by (iprover intro: terminatess-Fault)
  next
    case Stuck thus ?thesis
      by (iprover intro: terminatess-Stuck)
  qed
 qed
qed


ML ‹
  ML-Thms.bind-thm (rtrancl-induct3, Split-Rule.split-rule @{context}
    (Rule-Insts.read-instantiate @{context}
      [(((a, 0), Position.none), (ax, ay, az)),
       (((b, 0), Position.none), (bx, by, bz))] []
      @{thm rtranclp-induct}));
›

lemma steps-preserves-terminations:
  assumes steps: Γ⊢(cs,css,s) →* (cs',css',t)
  shows Γ⊢cs,css⇓s ⟹ Γ⊢cs',css'⇓t
using steps
proof (induct rule: rtrancl-induct3 [consumes 1])
  assume Γ⊢cs,css⇓s then show Γ⊢cs,css⇓s.
next
  fix cs'' css'' w cs' css' t
```

assume $\Gamma\vdash(cs'',css'',\ w) \rightarrow (cs',css',\ t)$ $\Gamma\vdash cs,css\Downarrow s \Longrightarrow \Gamma\vdash cs'',css''\Downarrow w$
$\Gamma\vdash cs,css\Downarrow s$
then show $\Gamma\vdash cs',css'\Downarrow t$
by (*blast dest*: *step-preserves-terminations*)
qed

theorem *steps-preserves-termination*:
assumes *steps*: $\Gamma\vdash([c],[],s) \rightarrow^* (c'\#cs',css',t)$
assumes *term-c*: $\Gamma\vdash c\downarrow s$
shows $\Gamma\vdash c'\downarrow t$
proof −
from *term-c* **have** $\Gamma\vdash[c],[]\Downarrow s$
by (*auto intro*: *terminatess.intros*)
from *steps this*
have $\Gamma\vdash c'\#cs',css'\Downarrow t$
by (*rule steps-preserves-terminations*)
thus $\Gamma\vdash c'\downarrow t$
by (*auto elim*: *terminatess-elim-cases*)
qed

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** $((\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i)\ i,\ f\ i) \in r$
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** $x=\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i$],*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 *not-inf-Fault′*:
assumes *enum-step*: $\forall\, i.\ \Gamma\vdash f\ i \to f\ (Suc\ i)$
shows $\bigwedge k\ cs.\ f\ k = (cs,css,Fault\ m) \Longrightarrow False$
proof (*induct css*)
case *Nil*
have *f-k*: $f\ k = (cs,[],Fault\ m)$ **by** *fact*
have $\bigwedge k.\ f\ k = (cs,[],Fault\ m) \Longrightarrow False$
proof (*induct cs*)
case *Nil*
have $f\ k = ([],\ [],\ Fault\ m)$ **by** *fact*
moreover
from *enum-step* **have** $\Gamma\vdash f\ k \to f\ (Suc\ k)$**..**
ultimately show *False*
by (*fastforce elim*: *step-elim-cases*)
next
case (*Cons c cs*)
have *fk*: $f\ k = (c\ \#\ cs,\ [],\ Fault\ m)$ **by** *fact*
from *enum-step* **have** $\Gamma\vdash f\ k \to f\ (Suc\ k)$**..**
with *fk* **have** $f\ (Suc\ k) = (cs,[],Fault\ m)$
by (*fastforce elim*: *step-elim-cases*)
with *enum-step Cons.hyps*
show *False*
by *blast*
qed
from *this f-k* **show** *False* **by** *blast*
next
case (*Cons ds css*)
then obtain *nrms abrs* **where** *ds*: *ds*=(*nrms*,*abrs*) **by** (*cases ds*) *auto*
have *hyp*: $\bigwedge k\ cs.\ f\ k = (cs,css,Fault\ m) \Longrightarrow False$ **by** *fact*
have $\bigwedge k.\ f\ k = (cs,(nrms,abrs)\#css,Fault\ m) \Longrightarrow False$
proof (*induct cs*)
case *Nil*
have *fk*: $f\ k = ([],\ (nrms,\ abrs)\ \#\ css,\ Fault\ m)$ **by** *fact*
from *enum-step* **have** $\Gamma\vdash f\ k \to f\ (Suc\ k)$**..**
with *fk* **have** $f\ (Suc\ k) = (nrms,css,Fault\ m)$
by (*fastforce elim*: *step-elim-cases*)
thus *?case*
by (*rule hyp*)
next
case (*Cons c cs*)
have *fk*: $f\ k = (c\#cs,\ (nrms,\ abrs)\ \#\ css,\ Fault\ m)$ **by** *fact*
have *hyp1*: $\bigwedge k.\ f\ k = (cs,\ (nrms,\ abrs)\ \#\ css,\ Fault\ m) \Longrightarrow False$ **by** *fact*
from *enum-step* **have** $\Gamma\vdash f\ k \to f\ (Suc\ k)$**..**

```
    with fk have f (Suc k) = (cs,(nrms,abrs)#css,Fault m)
      by (fastforce elim: step-elim-cases)
    thus ?case
      by (rule hyp1)
  qed
  with ds Cons.prems show False by auto
qed

lemma not-inf-Fault:
  ¬ inf Γ cs css (Fault m)
apply (rule not-infI)
apply (rule-tac f=f in not-inf-Fault')
by auto

lemma not-inf-Stuck':
  assumes enum-step: ∀ i. Γ⊢f i → f (Suc i)
  shows ⋀k cs. f k = (cs,css,Stuck) ⟹ False
proof (induct css)
  case Nil
  have f-k: f k = (cs,[],Stuck) by fact
  have ⋀k. f k = (cs,[],Stuck) ⟹ False
  proof (induct cs)
    case Nil
    have f k = ([], [], Stuck) by fact
    moreover
    from enum-step have Γ⊢f k → f (Suc k)..
    ultimately show False
      by (fastforce elim: step-elim-cases)
  next
    case (Cons c cs)
    have fk: f k = (c # cs, [], Stuck) by fact
    from enum-step have Γ⊢f k → f (Suc k)..
    with fk have f (Suc k) = (cs,[],Stuck)
      by (fastforce elim: step-elim-cases)
    with enum-step Cons.hyps
    show False
      by blast
  qed
  from this f-k show False .
next
  case (Cons ds css)
  then obtain nrms abrs where ds: ds=(nrms,abrs) by (cases ds) auto
  have hyp: ⋀k cs. f k = (cs,css,Stuck) ⟹ False by fact
  have ⋀k. f k = (cs,(nrms,abrs)#css,Stuck) ⟹ False
  proof (induct cs)
    case Nil
    have fk: f k = ([], (nrms, abrs) # css, Stuck) by fact
    from enum-step have Γ⊢f k → f (Suc k)..
    with fk have f (Suc k) = (nrms,css,Stuck)
```

    by (*fastforce elim*: *step-elim-cases*)
  thus *?case*
    by (*rule hyp*)
next
  case (*Cons c cs*)
  have *fk*: $f\ k = (c\#cs,\ (nrms,\ abrs)\ \#\ css,\ Stuck)$ **by** *fact*
  have *hyp1*: $\bigwedge k.\ f\ k = (cs,\ (nrms,\ abrs)\ \#\ css,\ Stuck) \Longrightarrow False$ **by** *fact*
  from *enum-step* **have** $\Gamma\vdash f\ k \to f\ (Suc\ k)$**..**
  with *fk* **have** $f\ (Suc\ k) = (cs,(nrms,abrs)\#css,Stuck)$
    by (*fastforce elim*: *step-elim-cases*)
  thus *?case*
    by (*rule hyp1*)
qed
with *ds Cons.prems* **show** *False* **by** *auto*
qed

lemma *not-inf-Stuck*:
  $\neg$ *inf* Γ *cs css Stuck*
apply (*rule not-infI*)
apply (*rule-tac f=f* **in** *not-inf-Stuck'*)
by *auto*

lemma *last-butlast-app*:
assumes *butlast*: *butlast as* = *xs* @ *butlast bs*
assumes *not-Nil*: *bs* ≠ [] *as* ≠ []
assumes *last*: *fst* (*last as*) = *fst* (*last bs*) *snd* (*last as*) = *snd* (*last bs*)
shows *as* = *xs* @ *bs*
proof −
  from *last* **have** *last as* = *last bs*
    by (*cases last as*,*cases last bs*) *simp*
  moreover
  from *not-Nil* **have** *as* = *butlast as* @ [*last as*] *bs* = *butlast bs* @ [*last bs*]
    by *auto*
  ultimately show *?thesis*
    using *butlast*
    by *simp*
qed

lemma *last-butlast-tl*:
assumes *butlast*: *butlast bs* = *x* # *butlast as*
assumes *not-Nil*: *bs* ≠ [] *as* ≠ []
assumes *last*: *fst* (*last as*) = *fst* (*last bs*) *snd* (*last as*) = *snd* (*last bs*)
shows *as* = *tl bs*
proof −
  from *last* **have** *last as* = *last bs*
    by (*cases last as*,*cases last bs*) *simp*
  moreover
  from *not-Nil* **have** *as* = *butlast as* @ [*last as*] *bs* = *butlast bs* @ [*last bs*]

```
    by auto
  ultimately show ?thesis
    using butlast
    by simp
qed

locale inf =
fixes CS:: ('s,'p,'f) config ⇒ ('s, 'p,'f) com list
  and CSS:: ('s,'p,'f) config ⇒ ('s, 'p,'f) continuation list
  and S:: ('s,'p,'f) config ⇒ ('s,'f) xstate
  defines CS-def : CS ≡ fst
  defines CSS-def : CSS ≡ λc. fst (snd c)
  defines S-def: S ≡ λc. snd (snd c)

lemma (in inf) steps-hd-drop-suffix:
assumes f-0: f 0 = (c#cs,css,s)
assumes f-step: ∀ i. Γ⊢ f(i) → f(Suc i)
assumes not-finished: ∀ i < k. ¬ (CS (f i) = cs ∧ CSS (f i) = css)
assumes simul: ∀ i≤k.
        (if pcss i = [] then CSS (f i)=css ∧ CS (f i)=pcs i@cs
                 else CS (f i)=pcs i ∧
                      CSS (f i)= butlast (pcss i)@
                              [(fst (last (pcss i))@cs,(snd (last (pcss i)))@cs)]@
                              css)
defines p≡λi. (pcs i, pcss i, S (f i))
shows ∀ i<k. Γ⊢ p i → p (Suc i)
using not-finished simul
proof (induct k)
  case 0
  thus ?case by simp
next
  case (Suc k)
  have simul: ∀ i≤Suc k.
        (if pcss i = [] then CSS (f i)=css ∧ CS (f i)=pcs i@cs
                 else CS (f i)=pcs i ∧
                      CSS (f i)= butlast (pcss i)@
                              [(fst (last (pcss i))@cs,(snd (last (pcss i)))@cs)]@
                              css) by fact
  have not-finished': ∀ i < Suc k. ¬ (CS (f i) = cs ∧ CSS (f i) = css) by fact
  with simul
  have not-finished: ∀ i<Suc k. ¬ (pcs i = [] ∧ pcss i = [])
    by (auto simp add: CS-def CSS-def S-def split: if-split-asm)
  show ?case
  proof (clarify)
    fix i
    assume i-le-Suc-k: i < Suc k
    show Γ⊢ p i → p (Suc i)
    proof (cases i < k)
      case True
```

with *not-finished′ simul Suc.hyps*
show *?thesis*
by *auto*
next
case *False*
with *i-le-Suc-k*
have *eq-i-k*: *i*=*k*
by *simp*
show Γ⊢*p i* → *p* (*Suc i*)
proof −
obtain *cs′ css′ t′* **where**
f-Suc-i: *f* (*Suc i*) = (*cs′*, *css′*, *t′*)
by (*cases f* (*Suc i*))
obtain *cs′′ css′′ t′′* **where**
f-i: *f i* = (*cs′′*,*css′′*,*t′′*)
by (*cases f i*)
from *not-finished eq-i-k*
have *pcs-pcss-not-Nil*: ¬ (*pcs i* = [] ∧ *pcss i* = [])
by *auto*
from *simul* [*rule-format*, *of i*] *i-le-Suc-k f-i*
have *pcs-pcss-i*:
if pcss i = [] *then css′′*=*css* ∧ *cs′′*=*pcs i*@*cs*
else cs′′=*pcs i* ∧
css′′= *butlast* (*pcss i*)@
[(*fst* (*last* (*pcss i*))@*cs*,(*snd* (*last* (*pcss i*)))@*cs*)]@
css
by (*simp add*: *CS-def CSS-def S-def cong*: *if-cong*)
from *simul* [*rule-format*, *of Suc i*] *i-le-Suc-k f-Suc-i*
have *pcs-pcss-Suc-i*:
if pcss (*Suc i*) = [] *then css′* = *css* ∧ *cs′* = *pcs* (*Suc i*) @ *cs*
else cs′ = *pcs* (*Suc i*) ∧
css′ = *butlast* (*pcss* (*Suc i*)) @
[(*fst* (*last* (*pcss* (*Suc i*))) @ *cs*, *snd* (*last* (*pcss* (*Suc i*))) @ *cs*)] @
css
by (*simp add*: *CS-def CSS-def S-def cong*: *if-cong*)
show *?thesis*
proof (*cases pcss i* = [])
case *True*
note *pcss-Nil* = *this*
with *pcs-pcss-i pcs-pcss-not-Nil* **obtain** *p ps* **where**
pcs-i: *pcs i* = *p*#*ps* **and**
css′′: *css′′*=*css* **and**
cs′′: *cs′′*=(*p*#*ps*)@*cs*
by (*auto simp add*: *neq-Nil-conv*)
with *f-i* **have** *f i* = (*p*#(*ps*@*cs*),*css*,*t′′*)
by *simp*
with *f-Suc-i f-step* [*rule-format*, *of i*]
have *step-css*: Γ⊢ (*p*#(*ps*@*cs*),*css*,*t′′*) → (*cs′*,*css′*,*t′*)
by *simp*

from *step-Cons'* [*OF this*, *of p ps@cs*]
obtain *css'''* **where**
  css''': *css'* = *css'''* @ *css*
  if css''' = [] then $\exists$ *p. cs'* = *p* @ *ps* @ *cs*
  else ($\exists$ *pnorm pabr. css'''*=[(*pnorm* @ *ps* @ *cs*,*pabr* @ *ps* @ *cs*)])
  by *auto*
show *?thesis*
proof (*cases css'''* = [])
  case *True*
  with *css'''*
  obtain *p'* **where**
    css': *css'* = *css* **and**
    cs': *cs'* = *p'* @ *ps* @ *cs*
    by *auto*

  from *css' cs' step-css*
  have *step*: $\Gamma\vdash$ (*p*#(*ps*@*cs*),*css*,*t''*) $\rightarrow$ (*p'*@*ps*@*cs*,*css*,*t'*)
    by *simp*
  hence $\Gamma\vdash$ ((*p*#*ps*)@*cs*,*css*,*t''*) $\rightarrow$ ((*p'*@*ps*)@*cs*,*css*,*t'*)
    by *simp*
  from *drop-suffix-css-step'* [*OF drop-suffix-same-css-step* [*OF this*],
    where *xs*=*css* **and** *css*=[] **and** *css'*=[]]
  have $\Gamma\vdash$ (*p*#*ps*,[],*t''*) $\rightarrow$ (*p'*@*ps*,[],*t'*)
    by *simp*
  moreover
  from *css' cs' pcs-pcss-Suc-i*
  obtain *pcs* (*Suc i*) = *p'*@*ps* **and** *pcss* (*Suc i*) = []
    by (*simp split*: *if-split-asm*)
  ultimately show *?thesis*
    using *pcs-i pcss-Nil f-i f-Suc-i*
    by (*simp add*: *CS-def CSS-def S-def p-def*)
next
  case *False*
  with *css'''*
  obtain *pnorm pabr* **where**
    css': *css'*=*css'''*@*css*
    css'''=[(*pnorm* @ *ps* @ *cs*,*pabr* @ *ps* @ *cs*)]
    by *auto*
  with *css''' step-css*
have $\Gamma\vdash$ (*p*#*ps*@*cs*,*css*,*t''*) $\rightarrow$ (*cs'*,[(*pnorm*@*ps*@*cs*,*pabr*@*ps*@*cs*)]@*css*,*t'*)
    by *simp*
  then
  have $\Gamma\vdash$(*p*#*ps*, *css*, *t''*) $\rightarrow$ (*cs'*, [(*pnorm*@*ps*, *pabr*@*ps*)] @ *css*, *t'*)
    by (*rule drop-suffix-hd-css-step*)
  from *drop-suffix-css-step'* [*OF this*,
    where *css*=[] **and** *xs*=*css* **and** *css'*=[(*pnorm*@*ps*, *pabr*@*ps*)]]
  have $\Gamma\vdash$ (*p*#*ps*,[],*t''*) $\rightarrow$ (*cs'*,[(*pnorm*@*ps*, *pabr*@*ps*)],*t'*)
    by *simp*
  moreover

from *css'* *pcs-pcss-Suc-i*
obtain *pcs* (*Suc i*) = *cs'* *pcss* (*Suc i*) = [(*pnorm*@*ps*, *pabr*@*ps*)]
apply (*cases pcss* (*Suc i*))
apply (*auto split*: *if-split-asm*)
done
ultimately show *?thesis*
using *pcs-i pcss-Nil f-i f-Suc-i*
by (*simp add*: *p-def CS-def CSS-def S-def*)
qed
next
case *False*
note *pcss-i-not-Nil* = *this*
with *pcs-pcss-i* **obtain**
cs'': *cs''*=*pcs i* **and**
css'': *css''*= *butlast* (*pcss i*)@
[(*fst* (*last* (*pcss i*))@*cs*,(*snd* (*last* (*pcss i*)))@*cs*)]@
css
by *auto*
from *f-Suc-i f-i f-step* [*rule-format*, *of i*]
have *step-i-full*: Γ⊢ (*cs''*,*css''*,*t''*) → (*cs'*,*css'*,*t'*)
by *simp*
show *?thesis*
proof (*cases cs''*)
case (*Cons c' cs*)
with *step-Cons'* [*OF step-i-full*]
obtain *css'''* **where** *css'*: *css'* = *css'''*@*css''*
by *auto*
with *step-i-full*
have Γ⊢ (*cs''*,*css''*,*t''*) → (*cs'*,*css'''*@*css''*,*t'*)
by *simp*
from *Cons-change-css-step* [*OF this*, **where** *xss*=*pcss i*] *Cons cs''*
have Γ⊢ (*pcs i*, *pcss i*,*t''*) → (*cs'*,*css'''*@*pcss i*,*t'*)
by *simp*
moreover
from *cs'' css'' css' False pcs-pcss-Suc-i*
obtain *pcs* (*Suc i*) = *cs'* *pcss* (*Suc i*) = *css'''*@*pcss i*
apply (*auto split*: *if-split-asm*)
apply (*drule* (*4*) *last-butlast-app*)
by *simp*
ultimately show *?thesis*
using *f-i f-Suc-i*
by (*simp add*: *p-def CS-def CSS-def S-def*)
next
case *Nil*
note *cs''-Nil* = *this*
show *?thesis*
proof (*cases butlast* (*pcss i*))
case (*Cons bpcs bpcss*)
with *cs''-Nil step-i-full css''*

have ∗: Γ⊢ ([],[*hd css''*]@*tl css''*,*t''*) → (*cs'*,*css'*,*t'*)
  by *simp*
moreover
from *step-Nil* [*OF* ∗]
have *css'*: *css'*=*tl css''*
  by *simp*
ultimately have
  step-i-full: Γ⊢ ([],[*hd css''*]@*tl css''*,*t''*) → (*cs'*,*tl css''*,*t'*)
  by *simp*
from *css'' Cons pcss-i-not-Nil*
have *hd css''* = *hd* (*pcss i*)
  by (*auto simp add*: *neq-Nil-conv split*: *if-split-asm*)
with *cs'' cs''-Nil*
  Nil-change-css-step [**where** *ass*=[*hd css''*] **and**
  css=*tl css''* **and** *ass'*=[] **and**
  xss=*tl* (*pcss i*), *simplified*, *OF step-i-full* [*simplified*]]
have Γ⊢ (*pcs i*,[*hd* (*pcss i*)]@*tl* (*pcss i*),*t''*) → (*cs'*,*tl* (*pcss i*),*t'*)
  by *simp*
with *pcss-i-not-Nil*
have Γ⊢ (*pcs i*,*pcss i*,*t''*) → (*cs'*,*tl* (*pcss i*),*t'*)
  by *simp*
moreover
from *css' css'' cs''-Nil Cons pcss-i-not-Nil pcs-pcss-Suc-i*
obtain *pcs* (*Suc i*) = *cs' pcss* (*Suc i*) = *tl* (*pcss i*)
  apply (*clarsimp split*: *if-split-asm*)
  apply (*drule* (*4*) *last-butlast-tl*)
  by *simp*
ultimately show *?thesis*
  using *f-i f-Suc-i*
  by (*simp add*: *p-def CS-def CSS-def S-def*)
next
case *Nil*
with *css'' pcss-i-not-Nil*
obtain *pnorm pabr*
  where *css''*: *css''*= [(*pnorm*@*cs*,*pabr*@*cs*)]@*css* **and**
  pcss-i: *pcss i* = [(*pnorm*,*pabr*)]
  by (*force simp add*: *neq-Nil-conv split*: *if-split-asm*)
with *cs''-Nil step-i-full*
have Γ⊢([],[(*pnorm*@*cs*,*pabr*@*cs*)]@*css*,*t''*) → (*cs'*,*css'*,*t'*)
  by *simp*
from *step-Nil* [*OF this*]
obtain
  css': *css'*=*css* **and**
  cs': (*case t'' of*
      Abrupt s' ⇒ *cs'* = *pabr* @ *cs* ∧ *t'* = *Normal s'*
    | - ⇒ *cs'* = *pnorm* @ *cs* ∧ *t'* = *t''*)
  by (*simp cong*: *xstate.case-cong*)
let *?pcs-Suc-i* = (*case t'' of Abrupt s'* ⇒ *pabr* | - ⇒ *pnorm*)
from *cs'*

have $\Gamma\vdash([],[(pnorm,pabr)],t'') \to (?pcs\text{-}Suc\text{-}i,[],t')$
by (*auto intro*: *step.intros split*: *xstate.splits*)
moreover
from *css'' css' cs' pcss-i pcs-pcss-Suc-i*
obtain $pcs\ (Suc\ i) = ?pcs\text{-}Suc\text{-}i\ pcss\ (Suc\ i) = []$
by (*simp split*: *if-split-asm xstate.splits*)
ultimately
show *?thesis*
using *pcss-i cs'' cs''-Nil f-i f-Suc-i*
by (*simp add*: *p-def CS-def CSS-def S-def*)
qed
qed
qed
qed
qed
qed
qed

lemma *k-steps-to-rtrancl*:
assumes *steps*: $\forall i<k.\ \Gamma\vdash\ p\ i \to p\ (Suc\ i)$
shows $\Gamma\vdash p\ 0\to^*\ p\ k$
using *steps*
proof (*induct k*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc k*)
have $\forall i<Suc\ k.\ \Gamma\vdash\ p\ i \to p\ (Suc\ i)$ **by** *fact*
then obtain
step-le-k: $\forall i<k.\ \Gamma\vdash\ p\ i \to p\ (Suc\ i)$ **and** *step-k*: $\Gamma\vdash\ p\ k \to p\ (Suc\ k)$
by *auto*
from *Suc.hyps* [*OF step-le-k*]
have $\Gamma\vdash\ p\ 0\ \to^*\ p\ k$**.**
also note *step-k*
finally show *?case* **.**
qed

lemma (**in** *inf*) *steps-hd-drop-suffix-finite*:
assumes *f-0*: $f\ 0 = (c\#cs,css,s)$
assumes *f-step*: $\forall i.\ \Gamma\vdash\ f(i) \to f(Suc\ i)$
assumes *not-finished*: $\forall i < k.\ \neg\ (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$
assumes *simul*: $\forall i\le k.$
$(if\ pcss\ i = []\ then\ CSS\ (f\ i)=css \wedge CS\ (f\ i)=pcs\ i@cs$
$else\ CS\ (f\ i)=pcs\ i\ \wedge$
$CSS\ (f\ i)=\ butlast\ (pcss\ i)@$
$[(fst\ (last\ (pcss\ i))@cs,(snd\ (last\ (pcss\ i)))@cs)]@$
$css)$
shows $\Gamma\vdash([c],[],s)\ \to^*\ (pcs\ k,\ pcss\ k,\ S\ (f\ k))$

proof −
  from *steps-hd-drop-suffix* [*OF f-0 f-step not-finished simul*]
  have ∀ *i*<*k*. Γ⊢ (*pcs i*, *pcss i*, *S* (*f i*)) →
        (*pcs* (*Suc i*), *pcss* (*Suc i*), *S* (*f* (*Suc i*)))**.**
  from *k-steps-to-rtrancl* [*OF this*]
  have Γ⊢ (*pcs 0*, *pcss 0*, *S* (*f 0*)) →* (*pcs k*, *pcss k*, *S* (*f k*))**.**
  moreover from *f-0 simul* [*rule-format*, *of 0*]
  have (*pcs 0*, *pcss 0*, *S* (*f 0*)) = ([*c*],[],*s*)
    by (*auto split*: *if-split-asm simp add*: *CS-def CSS-def S-def*)
  ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *inf*) *steps-hd-drop-suffix-infinite*:
assumes *f-0*: *f 0* = (*c#cs*,*css*,*s*)
assumes *f-step*: ∀ *i*. Γ⊢ *f*(*i*) → *f*(*Suc i*)
assumes *not-finished*: ∀ *i*. ¬ (*CS* (*f i*) = *cs* ∧ *CSS* (*f i*) = *css*)

assumes *simul*: ∀ *i*.
    (*if pcss i* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs i*@*cs*
        else CS (*f i*)=*pcs i* ∧
          CSS (*f i*)= *butlast* (*pcss i*)@
              [(*fst* (*last* (*pcss i*))@*cs*,(*snd* (*last* (*pcss i*)))@*cs*)]@
              css)
defines *p*≡λ*i*. (*pcs i*, *pcss i*, *S* (*f i*))
shows Γ⊢ *p i* → *p* (*Suc i*)
proof −
  from *steps-hd-drop-suffix* [*OF f-0 f-step*, *of Suc i pcss pcs*] *not-finished simul*
  show *?thesis*
    by (*auto simp add*: *p-def*)
qed

lemma (**in** *inf*) *steps-hd-progress*:
assumes *f-0*: *f 0* = (*c#cs*,*css*,*s*)
assumes *f-step*: ∀ *i*. Γ⊢ *f*(*i*) → *f*(*Suc i*)
assumes *c-unfinished*: ∀ *i* < *k*. ¬ (*CS* (*f i*) = *cs* ∧ *CSS* (*f i*) = *css*)
shows ∀ *i* ≤ *k*. (∃ *pcs pcss*.
    (*if pcss* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs*@*cs*
     else CS (*f i*)=*pcs* ∧
        CSS (*f i*)= *butlast pcss*@
            [(*fst* (*last pcss*)@*cs*,(*snd* (*last pcss*))@*cs*)]@
            css))
using *c-unfinished*
proof (*induct k*)
  case *0*
  with *f-0* **show** *?case*
    by (*simp add*: *CSS-def CS-def*)
next
  case (*Suc k*)
  have *c-unfinished*: ∀ *i*<*Suc k*. ¬ (*CS* (*f i*) = *cs* ∧ *CSS* (*f i*) = *css*) **by** *fact*

hence *c-unfinished'*: $\forall$ *i*< *k*. ¬ (*CS* (*f i*) = *cs* ∧ *CSS* (*f i*) = *css*) **by** *simp*
show *?case*
proof (*clarify*)
  fix *i*
  assume *i-le-Suc-k*: *i* ≤ *Suc k*
  show ∃ *pcs pcss*.
        (*if pcss* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs*@*cs*
         else CS (*f i*)=*pcs* ∧
              CSS (*f i*)= *butlast pcss*@
                         [(*fst* (*last pcss*)@*cs*,(*snd* (*last pcss*))@*cs*)]@
                         css)
  proof (*cases i* < *Suc k*)
    case *True*
    with *Suc.hyps* [*OF c-unfinished'*, *rule-format*, *of i*] *c-unfinished*
    show *?thesis*
      by *auto*
  next
    case *False*
    with *i-le-Suc-k* **have** *eq-i-Suc-k*: *i*=*Suc k*
      by *auto*
    obtain *cs' css' t'* **where**
      f-Suc-k: *f* (*Suc k*) = (*cs'*, *css'*, *t'*)
      by (*cases f* (*Suc k*))
    obtain *cs'' css'' t''* **where**
      f-k: *f k* = (*cs''*,*css''*,*t''*)
      by (*cases f k*)
    with *Suc.hyps* [*OF c-unfinished'*,*rule-format*, *of k*]
    obtain *pcs pcss* **where**
      pcs-pcss-k:
      if pcss = [] *then css''* = *css* ∧ *cs''* = *pcs* @ *cs*
       else cs'' = *pcs* ∧
            css'' = *butlast pcss* @
                         [(*fst* (*last pcss*) @ *cs*, *snd* (*last pcss*) @ *cs*)] @
                         css
      by (*auto simp add*: *CSS-def CS-def cong*: *if-cong*)
    from *c-unfinished* [*rule-format*, *of k*] *f-k pcs-pcss-k*
    have *pcs-pcss-empty*: ¬ (*pcs* = [] ∧ *pcss* = [])
      by (*auto simp add*: *CS-def CSS-def S-def split*: *if-split-asm*)
    show *?thesis*
    proof (*cases pcss* = [])
      case *True*
      note *pcss-Nil* = *this*
      with *pcs-pcss-k pcs-pcss-empty* **obtain** *p ps* **where**
        pcs-i: *pcs* = *p*#*ps* **and**
        css'': *css''*=*css* **and**
        cs'': *cs''*=(*p*#*ps*)@*cs*
        by (*cases pcs*) *auto*
      with *f-k* **have** *f k* = (*p*#(*ps*@*cs*),*css*,*t''*)
        by *simp*

with *f-Suc-k f-step* [*rule-format*, *of k*]
have *step-css*: $\Gamma\vdash$ $(p\#(ps@cs),css,t'') \rightarrow (cs',css',t')$
  by *simp*
from *step-Cons'* [*OF this*, *of p ps@cs*]
obtain *css'''* **where**
  css''': $css' = css'''$ @ css
  if $css''' = []$ *then* $\exists\, p.\ cs' = p$ @ ps @ cs
  else $(\exists\, pnorm\ pabr.\ css'''=[(pnorm$ @ ps @ $cs,pabr$ @ ps @ $cs)])$
  by *auto*
show *?thesis*
proof (*cases* $css''' = []$)
  case *True*
  with *css'''*
  obtain p' **where**
    css': $css' = css$ **and**
    cs': $cs' = p'$ @ ps @ cs
    by *auto*
  from *css' cs' f-Suc-k*
  show *?thesis*
    apply (*rule-tac* $x=p'@ps$ **in** *exI*)
    apply (*rule-tac* $x=[]$ **in** *exI*)
    apply (*simp add*: *CSS-def CS-def eq-i-Suc-k*)
    done
next
  case *False*
  with *css'''*
  obtain *pnorm pabr* **where**
    css': $css'=css'''@css$
    $css'''=[(pnorm$ @ ps @ $cs,pabr$ @ ps @ $cs)]$
    by *auto*
  with *f-Suc-k eq-i-Suc-k*
  show *?thesis*
    apply (*rule-tac* $x=cs'$ **in** *exI*)
    apply (*rule-tac* $x=[(pnorm@ps,\ pabr@ps)]$ **in** *exI*)
    by (*simp add*: *CSS-def CS-def*)
qed
next
case *False*
note *pcss-k-not-Nil* = *this*
with *pcs-pcss-k* **obtain**
  cs'': $cs''=pcs$ **and**
  css'': $css''=$ *butlast pcss*@
      $[(fst\ (last\ pcss)@cs,(snd\ (last\ pcss))@cs)]@$
      css
  by *auto*
from *f-Suc-k f-k f-step* [*rule-format*, *of k*]
have *step-i-full*: $\Gamma\vdash$ $(cs'',css'',t'') \rightarrow (cs',css',t')$
  by *simp*
show *?thesis*

proof (*cases cs''*)
  case (*Cons c' cs*)
  with *step-Cons'* [*OF step-i-full*]
  obtain *css'''* **where** *css'*: *css' = css'''@css''*
    by *auto*
  with *cs'' css'' f-Suc-k eq-i-Suc-k pcss-k-not-Nil*
  show *?thesis*
    apply (*rule-tac x=cs'* **in** *exI*)
    apply (*rule-tac x=css'''@pcss* **in** *exI*)
    by (*clarsimp simp add*: *CSS-def CS-def butlast-append*)
next
  case *Nil*
  note *cs''-Nil = this*
  show *?thesis*
  proof (*cases butlast pcss*)
    case (*Cons bpcs bpcss*)
    with *cs''-Nil step-i-full css''*
    have ∗: Γ⊢ ([],[*hd css''*]@*tl css''*,*t''*) → (*cs'*,*css'*,*t'*)
      by *simp*
    moreover
    from *step-Nil* [*OF* ∗]
    obtain *css'*: *css'=tl css''* **and**
          cs': *cs' = (case t'' of Abrupt s' ⇒ snd (hd css'')*
                        | - ⇒ fst (hd css''))
      by (*auto split*: *xstate.splits*)
    from *css'' Cons pcss-k-not-Nil*
    have *hd css'' = hd pcss*
      by (*auto simp add*: *neq-Nil-conv split*: *if-split-asm*)
    with *css' cs' css'' cs''-Nil Cons pcss-k-not-Nil f-Suc-k eq-i-Suc-k*
    show *?thesis*
      apply (*rule-tac x=cs'* **in** *exI*)
      apply (*rule-tac x=tl pcss* **in** *exI*)
      apply (*clarsimp split*: *xstate.splits*
              simp add: *CS-def CSS-def neq-Nil-conv split*: *if-split-asm*)
      done
  next
    case *Nil*
    with *css'' pcss-k-not-Nil*
    obtain *pnorm pabr*
      where *css''*: *css''*= [(*pnorm*@*cs*,*pabr*@*cs*)]@*css* **and**
      pcss-k: *pcss* = [(*pnorm*,*pabr*)]
      by (*force simp add*: *neq-Nil-conv split*: *if-split-asm*)
    with *cs''-Nil step-i-full*
    have Γ⊢([],[(*pnorm*@*cs*,*pabr*@*cs*)]@*css*,*t''*) → (*cs'*,*css'*,*t'*)
      by *simp*
    from *step-Nil* [*OF this*]
    obtain
      css': *css'=css* **and**
      cs': (*case t'' of*

Abrupt s' ⇒ cs' = pabr @ cs ∧ t' = Normal s'
| - ⇒ cs' = pnorm @ cs ∧ t' = t'')
by (*simp cong*: *xstate.case-cong*)
let *?pcs-Suc-k* = (*case t'' of Abrupt s' ⇒ pabr | - ⇒ pnorm*)
from *css'' css' cs' pcss-k f-Suc-k eq-i-Suc-k*
show *?thesis*
apply (*rule-tac x=?pcs-Suc-k* **in** *exI*)
apply (*rule-tac x=*[] **in** *exI*)
apply (*simp split*: *xstate.splits add*: *CS-def CSS-def*)
done
qed
qed
qed
qed
qed
qed

lemma (**in** *inf*) *inf-progress*:
assumes *f-0*: *f 0* = (*c#cs*,*css*,*s*)
assumes *f-step*: ∀ *i*. Γ⊢ *f*(*i*) → *f*(*Suc i*)
assumes *unfinished*: ∀ *i*. ¬ ((*CS* (*f i*) = *cs*) ∧ (*CSS* (*f i*) = *css*))
shows ∃ *pcs pcss*.
(*if pcss* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs*@*cs*
else CS (*f i*)=*pcs* ∧
CSS (*f i*)= *butlast pcss*@
[(*fst* (*last pcss*)@*cs*,(*snd* (*last pcss*))@*cs*)]@
css)
proof −
from *steps-hd-progress* [*OF f-0 f-step*, *of i*] *unfinished*
show *?thesis*
by *auto*
qed

lemma *skolemize1*: ∀ *x*. *P x* ⟶ (∃ *y*. *Q x y*) ⟹ ∃ *f*.∀ *x*. *P x* ⟶ *Q x* (*f x*)
by (*rule choice*) *blast*

lemma *skolemize2*: ∀ *x*. *P x* ⟶ (∃ *y z*. *Q x y z*) ⟹ ∃ *f g*.∀ *x*. *P x* ⟶ *Q x* (*f x*) (*g x*)
apply (*drule skolemize1*)
apply (*erule exE*)
apply (*drule skolemize1*)
apply *fast*
done

lemma *skolemize2'*: ∀ *x*.∃ *y z*. *P x y z* ⟹ ∃ *f g*.∀ *x*. *P x* (*f x*) (*g x*)
apply (*drule choice*)
apply (*erule exE*)
apply (*drule choice*)
apply *fast*

done

theorem (**in** *inf*) *inf-cases*:
  fixes c::($'s$,$'p$,$'f$) *com*
  assumes *inf*: *inf* Γ (c#cs) *css* s
  shows *inf* Γ [c] [] s $\vee$ ($\exists\, t$. $\Gamma\vdash\langle c,s\rangle \Rightarrow t \wedge$ *inf* Γ *cs css* t)
proof –
  from *inf* **obtain** f **where**
    f-0: $f\ 0$ = (c#cs,*css*,s) **and**
    f-step: ($\forall\, i$. $\Gamma\vdash f\ i \rightarrow f$ (*Suc* i))
    by (*auto simp add*: *inf-def*)
  show *?thesis*
  proof (*cases* $\exists\, i$. *CS* ($f\ i$) = *cs* $\wedge$ *CSS* ($f\ i$) = *css*)
    case *True*
    define k **where** k = (*LEAST* i. *CS* ($f\ i$) = *cs* $\wedge$ *CSS* ($f\ i$) = *css*)
    from *True*
    obtain *CS-f-k*: *CS* ($f\ k$) = *cs* **and** *CSS-f-k*: *CSS* ($f\ k$) = *css*
      apply –
      apply (*erule exE*)
      apply (*drule LeastI*)
      apply (*simp add*: *k-def*)
      done
    have *less-k-prop*: $\forall\, i<k$. $\neg$ (*CS* ($f\ i$) = *cs* $\wedge$ *CSS* ($f\ i$) = *css*)
      apply (*intro allI impI*)
      apply (*unfold k-def*)
      apply (*drule not-less-Least*)
      apply *simp*
      done
    have $\Gamma\vdash$([c], [], s) $\rightarrow^*$ ([],[],*S* ($f\ k$))
    proof –
      have $\forall\, i\le k$. $\exists$ *pcs pcss*.
        (*if pcss* = [] *then CSS* ($f\ i$)=*css* $\wedge$ *CS* ($f\ i$)=*pcs*@*cs*
          else CS ($f\ i$)=*pcs* $\wedge$
                CSS ($f\ i$)= *butlast pcss*@
                      [(*fst* (*last pcss*)@*cs*,(*snd* (*last pcss*))@*cs*)]@
                      css)
        by (*rule steps-hd-progress*
        [*OF f-0 f-step*, **where** k=k, *OF less-k-prop*])
      from *skolemize2* [*OF this*] **obtain** *pcs pcss* **where**
        pcs-pcss:
          $\forall\, i\le k$.
            (*if pcss* i = [] *then CSS* ($f\ i$)=*css* $\wedge$ *CS* ($f\ i$)=*pcs* i@*cs*
             else CS ($f\ i$)=*pcs* i $\wedge$
                CSS ($f\ i$)= *butlast* (*pcss* i)@
                      [(*fst* (*last* (*pcss* i))@*cs*,(*snd* (*last* (*pcss* i)))@*cs*)]@
                      css)
        by *iprover*
      from *pcs-pcss* [*rule-format*, *of* k] *CS-f-k CSS-f-k*
      have *finished*: *pcs* k = [] *pcss* k = []

by (*auto simp add*: *CS-def CSS-def S-def split*: *if-split-asm*)
from *pcs-pcss*
have *simul*: $\forall\, i \le k$. (*if pcss i* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs i*@*cs*
else CS (*f i*)=*pcs i* ∧
CSS (*f i*)= *butlast* (*pcss i*)@
[(*fst* (*last* (*pcss i*))@*cs*,(*snd* (*last* (*pcss i*)))@*cs*)]@
css)
by *auto*
from *steps-hd-drop-suffix-finite* [*OF f-0 f-step less-k-prop simul*] *finished*
show *?thesis*
by *simp*
qed
hence $\Gamma\vdash\langle c,s\rangle \Rightarrow S\ (f\ k)$
by (*rule steps-impl-exec*)
moreover
from *CS-f-k CSS-f-k f-step*
have *inf* Γ *cs css* (*S* (*f k*))
apply (*simp add*: *inf-def*)
apply (*rule-tac* $x{=}\lambda i.\ f\ (i + k)$ **in** *exI*)
apply *simp*
apply (*auto simp add*: *CS-def CSS-def S-def*)
done
ultimately
have ($\exists\, t$. $\Gamma\vdash\langle c,s\rangle \Rightarrow t$ ∧ *inf* Γ *cs css t*)
by *blast*
thus *?thesis*
by *simp*
next
case *False*
hence *unfinished*: $\forall\, i$. ¬ ((*CS* (*f i*) = *cs*) ∧ (*CSS* (*f i*) = *css*))
by *simp*
from *inf-progress* [*OF f-0 f-step this*]
have $\forall\, i$. ∃ *pcs pcss*.
(*if pcss* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs*@*cs*
else CS (*f i*)=*pcs* ∧
CSS (*f i*)= *butlast pcss*@
[(*fst* (*last pcss*)@*cs*,(*snd* (*last pcss*))@*cs*)]@
css)
by *auto*
from *skolemize2'* [*OF this*] **obtain** *pcs pcss* **where**
pcs-pcss: $\forall\, i$.
(*if pcss i* = [] *then CSS* (*f i*)=*css* ∧ *CS* (*f i*)=*pcs i*@*cs*
else CS (*f i*)=*pcs i* ∧
CSS (*f i*)= *butlast* (*pcss i*)@
[(*fst* (*last* (*pcss i*))@*cs*,(*snd* (*last* (*pcss i*)))@*cs*)]@
css)
by *iprover*
define *g* **where** *g i* = (*pcs i*, *pcss i*, *S* (*f i*)) **for** *i*
from *pcs-pcss* [*rule-format*, *of 0*] *f-0*

```
    have g 0 = ([c],[],s)
      by (auto split: if-split-asm simp add: CS-def CSS-def S-def g-def)
    moreover
    from steps-hd-drop-suffix-infinite [OF f-0 f-step unfinished pcs-pcss]
    have ∀ i. Γ⊢g i → g (Suc i)
      by (simp add: g-def)
    ultimately
    have inf Γ [c] [] s
      by (auto simp add: inf-def)
    thus ?thesis
      by simp
  qed
qed

lemma infE [consumes 1]:
  assumes inf: inf Γ (c#cs) css s
  assumes cases: inf Γ [c] [] s ⟹ P
                 ⋀t. ⟦Γ⊢⟨c,s⟩ ⇒ t; inf Γ cs css t⟧ ⟹ P
  shows P
using inf cases
apply −
apply (drule inf.inf-cases)
apply auto
done

lemma inf-Seq:
  inf Γ (Seq c1 c2#cs) css (Normal s) = inf Γ (c1#c2#cs) css (Normal s)
proof
  assume inf Γ (Seq c1 c2 # cs) css (Normal s)
  then obtain f where
    f-0: f 0 = (Seq c1 c2#cs,css,Normal s) and
    f-step: ∀ i. Γ⊢f i → f (Suc i)
    by (auto simp add: inf-def)
  from f-step [rule-format, of 0] f-0
  have f 1 = (c1#c2#cs,css,Normal s)
    by (auto elim: step-Normal-elim-cases)
  with f-step show inf Γ (c1#c2#cs) css (Normal s)
    apply (simp add: inf-def)
    apply (rule-tac x=λi. f (Suc i) in exI)
    apply simp
    done
next
  assume inf Γ (c1 # c2 # cs) css (Normal s)
  then obtain f where
    f-0: f 0 = (c1# c2#cs,css,Normal s) and
    f-step: ∀ i. Γ⊢f i → f (Suc i)
    by (auto simp add: inf-def)
  define g where g i = (case i of 0 ⇒ (Seq c1 c2#cs,css,Normal s) | Suc j ⇒ f
j) for i
```

with *f-0* **have**
Γ⊢g 0 → g (Suc 0)
by (*auto intro*: *step.intros*)
moreover
from *f-step* **have** ∀ i. i≠0 ⟶ Γ⊢g i → g (Suc i)
by (*auto simp add*: *g-def split*: *nat.splits*)
ultimately
show *inf* Γ (*Seq c1 c2 # cs*) *css* (*Normal s*)
apply (*simp add*: *inf-def*)
apply (*rule-tac x=g* **in** *exI*)
apply (*simp add*: *g-def split*: *nat.splits*)
done
qed

lemma *inf-WhileTrue*:
assumes *b*: s ∈ b
shows *inf* Γ (*While b c#cs*) *css* (*Normal s*) =
inf Γ (*c#While b c#cs*) *css* (*Normal s*)
proof
assume *inf* Γ (*While b c # cs*) *css* (*Normal s*)
then obtain *f* **where**
f-0: f 0 = (*While b c#cs,css,Normal s*) **and**
f-step: ∀ i. Γ⊢f i → f (Suc i)
by (*auto simp add*: *inf-def*)
from *b f-step* [*rule-format*, *of 0*] *f-0*
have f 1 = (*c#While b c#cs,css,Normal s*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* **show** *inf* Γ (*c # While b c # cs*) *css* (*Normal s*)
apply (*simp add*: *inf-def*)
apply (*rule-tac x=λi. f (Suc i)* **in** *exI*)
apply *simp*
done
next
assume *inf* Γ (*c # While b c # cs*) *css* (*Normal s*)
then obtain *f* **where**
f-0: f 0 = (*c # While b c #cs,css,Normal s*) **and**
f-step: ∀ i. Γ⊢f i → f (Suc i)
by (*auto simp add*: *inf-def*)
define *h* **where** h i = (*case i of 0* ⇒ (*While b c#cs,css,Normal s*) | *Suc j* ⇒ f
j) **for** *i*
with *b f-0* **have**
Γ⊢h 0 → h (Suc 0)
by (*auto intro*: *step.intros*)
moreover
from *f-step* **have** ∀ i. i≠0 ⟶ Γ⊢h i → h (Suc i)
by (*auto simp add*: *h-def split*: *nat.splits*)
ultimately
show *inf* Γ (*While b c # cs*) *css* (*Normal s*)
apply (*simp add*: *inf-def*)

apply (*rule-tac x=h* **in** *exI*)
apply (*simp add*: *h-def split*: *nat.splits*)
done
qed

lemma *inf-Catch*:
inf Γ (*Catch c1 c2#cs*) *css* (*Normal s*) = *inf* Γ [*c1*] ((*cs*,*c2#cs*)#*css*) (*Normal s*)
proof
assume *inf* Γ (*Catch c1 c2#cs*) *css* (*Normal s*)
then obtain *f* **where**
f-0: *f 0* = (*Catch c1 c2#cs*,*css*,*Normal s*) **and**
f-step: ∀ *i*. Γ⊢*f i* → *f* (*Suc i*)
by (*auto simp add*: *inf-def*)
from *f-step* [*rule-format*, *of 0*] *f-0*
have *f 1* = ([*c1*],(*cs*,*c2#cs*)#*css*,*Normal s*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* **show** *inf* Γ [*c1*] ((*cs*,*c2#cs*)#*css*) (*Normal s*)
apply (*simp add*: *inf-def*)
apply (*rule-tac x*=λ*i*. *f* (*Suc i*) **in** *exI*)
apply *simp*
done
next
assume *inf* Γ [*c1*] ((*cs*,*c2#cs*)#*css*) (*Normal s*)
then obtain *f* **where**
f-0: *f 0* = ([*c1*],(*cs*,*c2#cs*)#*css*,*Normal s*) **and**
f-step: ∀ *i*. Γ⊢*f i* → *f* (*Suc i*)
by (*auto simp add*: *inf-def*)
define *h* **where** *h i* = (*case i of 0* ⇒ (*Catch c1 c2#cs*,*css*,*Normal s*) | *Suc j* ⇒ *f j*) **for** *i*
with *f-0* **have**
Γ⊢*h 0* → *h* (*Suc 0*)
by (*auto intro*: *step.intros*)
moreover
from *f-step* **have** ∀ *i*. *i*≠*0* ⟶ Γ⊢*h i* → *h* (*Suc i*)
by (*auto simp add*: *h-def split*: *nat.splits*)
ultimately
show *inf* Γ (*Catch c1 c2 # cs*) *css* (*Normal s*)
apply (*simp add*: *inf-def*)
apply (*rule-tac x=h* **in** *exI*)
apply (*simp add*: *h-def split*: *nat.splits*)
done
qed

theorem *terminates-impl-not-inf*:
assumes *termi*: Γ⊢*c* ↓ *s*
shows ¬*inf* 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
    have f (Suc 0) = ([],[],Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  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
    have f (Suc 0) = ([],[],Normal (g s))
      by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
      by (auto elim: 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)
    with f-step [of 0]
    have Γ⊢([Spec r], [], Normal s) → f (Suc 0)
      by simp
    then show False
    proof (cases)
      fix t
      assume (s, t) ∈ r f (Suc 0) = ([], [], Normal t)
      with f-step [of 1]
      show False
        by (auto elim: step-elim-cases)
    next
      assume ∀ t. (s, t) ∉ r f (Suc 0) = ([], [], Stuck)
      with f-step [of 1]
      show False
        by (auto elim: step-elim-cases)
    qed
```

```
  qed
next
  case (Guard s g c m)
  have g: s ∈ g by fact
  have hyp: ¬ inf 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 g f-step [of 0] f-0
    have f (Suc 0) = ([c],[],Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have inf 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
    have f (Suc 0) = ([],[],Fault m)
      by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  qed
next
  case (Fault c m)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = ([c], [], Fault m)
    from f-step [of 0] f-0
    have f (Suc 0) = ([],[],Fault m)
      by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  qed
```

next
  case (*Seq c1 s c2*)
  have *hyp-c1*: $\neg$ *inf* Γ [*c1*] [] (*Normal s*) **by** *fact*
  have *hyp-c2*: $\forall s'.\ \Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow s' \longrightarrow \Gamma\vdash c2 \downarrow s' \wedge \neg$ *inf* Γ [*c2*] [] s' **by** *fact*
  have $\neg$ *inf* Γ ([*c1*,*c2*]) [] (*Normal s*)
  proof
    assume *inf* Γ [*c1*, *c2*] [] (*Normal s*)
    then show *False*
    proof (*cases rule*: *infE*)
      assume *inf* Γ [*c1*] [] (*Normal s*)
      with *hyp-c1* **show** *?thesis* **by** *simp*
    next
      fix *t*
      assume $\Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow t$ *inf* Γ [*c2*] [] *t*
      with *hyp-c2* **show** *?thesis* **by** *simp*
    qed
  qed
  thus *?case*
    by (*simp add*: *inf-Seq*)
next
  case (*CondTrue s b c1 c2*)
  have *b*: $s \in b$ **by** *fact*
  have *hyp-c1*: $\neg$ *inf* Γ [*c1*] [] (*Normal s*) **by** *fact*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = ([Cond\ b\ c1\ c2], [], Normal\ s)$
    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 *inf* Γ [*c1*] [] (*Normal s*)
      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*: $\neg$ *inf* Γ [*c2*] [] (*Normal s*) **by** *fact*
  show *?case*
  proof (*rule not-infI*)
    fix *f*
    assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
    assume *f-0*: $f\ 0 = ([Cond\ b\ c1\ c2], [], Normal\ s)$
    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 $inf\ \Gamma\ [c2]\ []\ (Normal\ s)$
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*: $s \in b$ **by** *fact*
have *hyp-c*: $\neg\ inf\ \Gamma\ [c]\ []\ (Normal\ s)$ **by** *fact*
have *hyp-w*: $\forall s'.\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow s' \longrightarrow$
$\Gamma\vdash While\ b\ c \downarrow s' \wedge \neg\ inf\ \Gamma\ [While\ b\ c]\ []\ s'$ **by** *fact*
have $\neg\ inf\ \Gamma\ [c,While\ b\ c]\ []\ (Normal\ s)$
proof
assume $inf\ \Gamma\ [c,While\ b\ c]\ []\ (Normal\ s)$
from *this hyp-c hyp-w* **show** *False*
by (*cases rule*: *infE*) *auto*
qed
with *b* **show** *?case*
by (*simp add*: *inf-WhileTrue*)
next
case (*WhileFalse s b c*)
have *b*: $s \notin b$ **by** *fact*
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f-0*: $f\ 0 = ([While\ b\ c],\ [],\ Normal\ s)$
from *b f-step* [*of 0*] *f-0*
have $f\ (Suc\ 0) = ([],[],Normal\ s)$
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*of 1*]
show *False*
by (*auto elim*: *step-elim-cases*)
qed
next
case (*Call p bdy s*)
have *bdy*: $\Gamma\ p = Some\ bdy$ **by** *fact*
have *hyp*: $\neg\ inf\ \Gamma\ [bdy]\ []\ (Normal\ s)$ **by** *fact*
have *not-inf-bdy*:
$\neg\ inf\ \Gamma\ [bdy]\ [([],[Throw])]\ (Normal\ s)$
proof
assume $inf\ \Gamma\ [bdy]\ [([],[Throw])]\ (Normal\ s)$
then show *False*
proof (*rule infE*)
assume $inf\ \Gamma\ [bdy]\ []\ (Normal\ s)$

with *hyp* **show** *False* **by** *simp*
next
fix t
assume $\Gamma\vdash\langle bdy,Normal\ s\rangle \Rightarrow t$
assume *inf*: *inf* Γ [] [([], [*Throw*])] t
then obtain f **where**
f-0: $f\ 0$ = ([],[([], [*Throw*])],t) **and**
f-step: $\forall\, i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
by (*auto simp add*: *inf-def*)
show *False*
proof (*cases t*)
case (*Normal t'*)
with *f-0 f-step* [*rule-format*, *of 0*]
have $f\ (Suc\ 0)$ = ([],[],(*Normal t'*))
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of Suc 0*]
show *False*
by (*auto elim*: *step.cases*)
next
case (*Abrupt t'*)
with *f-0 f-step* [*rule-format*, *of 0*]
have $f\ (Suc\ 0)$ = ([*Throw*],[],(*Normal t'*))
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of Suc 0*]
have $f\ (Suc\ (Suc\ 0))$ = ([],[],(*Abrupt t'*))
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of Suc*(*Suc 0*)]
show *False*
by (*auto elim*: *step.cases*)
next
case (*Fault m*)
with *f-0 f-step* [*rule-format*, *of 0*]
have $f\ (Suc\ 0)$ = ([],[],*Fault m*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of 1*]
have $f\ (Suc\ (Suc\ 0))$ = ([],[],*Fault m*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of Suc* (*Suc 0*)]
show *False*
by (*auto elim*: *step.cases*)
next
case *Stuck*
with *f-0 f-step* [*rule-format*, *of 0*]
have $f\ (Suc\ 0)$ = ([],[],*Stuck*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of 1*]
have $f\ (Suc\ (Suc\ 0))$ = ([],[],*Stuck*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of Suc* (*Suc 0*)]

```
        show False
          by (auto elim: step.cases)
      qed
    qed
  qed
  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 (Suc 0) =
          ([bdy],[([],[Throw])],Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have inf Γ [bdy] [([],[Throw])] (Normal s)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with not-inf-bdy
    show False by simp
  qed
next
  case (CallUndefined p s)
  have undef: 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 undef f-step [of 0] f-0
    have f (Suc 0) = ([],[],Stuck)
      by (auto elim: step-Normal-elim-cases)
    with f-step [rule-format, of Suc 0]
    show False
      by (auto elim: step-elim-cases)
  qed
next
  case (Stuck c)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = ([c], [], Stuck)
    from f-step [of 0] f-0
    have f (Suc 0) = ([],[],Stuck)
      by (auto elim: step-elim-cases)
    with f-step [rule-format, of Suc 0]
    show False
```

```
      by (auto elim: step-elim-cases)
  qed
next
  case (DynCom c s)
  have hyp: ¬ inf 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 inf 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: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = ([Throw], [], Normal s)
    from f-step [of 0] f-0
    have f (Suc 0) = ([],[],Abrupt s)
      by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  qed
next
  case (Abrupt c s)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = ([c], [], Abrupt s)
    from f-step [of 0] f-0
    have f (Suc 0) = ([],[],Abrupt s)
      by (auto elim: step-elim-cases)
    with f-step [rule-format, of Suc 0]
    show False
      by (auto elim: step-elim-cases)
  qed
next
```

case (*Catch c1 s c2*)
have *hyp-c1*: $\neg$ *inf* Γ [*c1*] [] (*Normal s*) **by** *fact*
have *hyp-c2*: $\forall s'$. $\Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow Abrupt\ s' \longrightarrow$
$\Gamma\vdash c2 \downarrow Normal\ s' \wedge \neg$ *inf* Γ [*c2*] [] (*Normal s'*) **by** *fact*
have $\neg$ *inf* Γ [*c1*] [([],[*c2*])] (*Normal s*)
proof
  assume *inf* Γ [*c1*] [([],[*c2*])] (*Normal s*)
  then show *False*
  proof (*rule infE*)
    assume *inf* Γ [*c1*] [] (*Normal s*)
    with *hyp-c1* **show** *False* **by** *simp*
  next
    fix *t*
    assume *eval*: $\Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow t$
    assume *inf*: *inf* Γ [] [([], [*c2*])] *t*
    then obtain *f* **where**
      f-0: *f 0* = ([],[([], [*c2*])],*t*) **and**
      f-step: $\forall i$. $\Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
      by (*auto simp add*: *inf-def*)
    show *False*
    proof (*cases t*)
      case (*Normal t'*)
      with *f-0 f-step* [*rule-format*, *of 0*]
      have *f* (*Suc 0*) = ([],[],*Normal t'*)
        by (*auto elim*: *step-Normal-elim-cases*)
      with *f-step* [*rule-format*, *of 1*]
      show *False*
        by (*auto elim*: *step-elim-cases*)
    next
      case (*Abrupt t'*)
      with *f-0 f-step* [*rule-format*, *of 0*]
      have *f* (*Suc 0*) = ([*c2*],[],*Normal t'*)
        by (*auto elim*: *step-Normal-elim-cases*)
      with *f-step eval Abrupt*
      have *inf* Γ [*c2*] [] (*Normal t'*)
        apply (*simp add*: *inf-def*)
        apply (*rule-tac x*=λi. *f* (*Suc i*) **in** *exI*)
        by *simp*
      with *eval hyp-c2 Abrupt* **show** *False* **by** *simp*
    next
      case (*Fault m*)
      with *f-0 f-step* [*rule-format*, *of 0*]
      have *f* (*Suc 0*) = ([],[],*Fault m*)
        by (*auto elim*: *step-Normal-elim-cases*)
      with *f-step* [*rule-format*, *of 1*]
      show *False*
        by (*auto elim*: *step-elim-cases*)
    next
      case *Stuck*

with *f-0 f-step* [*rule-format*, *of 0*]
have *f* (*Suc 0*) = ([],[],*Stuck*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* [*rule-format*, *of 1*]
show *False*
by (*auto elim*: *step-elim-cases*)
qed
qed
qed
thus *?case*
by (*simp add*: *inf-Catch*)
qed

lemma *terminatess-impl-not-inf*:
assumes *termi*: $\Gamma\vdash cs,css\Downarrow s$
shows $\neg inf\ \Gamma\ cs\ css\ s$
using *termi*
proof (*induct*)
case (*Nil s*)
show *?case*
proof (*rule not-infI*)
fix *f*
assume $\bigwedge i.\ \Gamma\vdash f\ i \to f\ (Suc\ i)$
hence $\Gamma\vdash f\ 0 \to f\ (Suc\ 0)$
by *simp*
moreover
assume *f 0* = ([], [], *s*)
ultimately show *False*
by (*fastforce elim*: *step.cases*)
qed
next
case (*ExitBlockNormal nrms css s abrs*)
have *hyp*: ¬ *inf* Γ *nrms css* (*Normal s*) **by** *fact*
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \to f\ (Suc\ i)$
assume *f0*: *f 0* = ([], (*nrms*, *abrs*) # *css*, *Normal s*)
with *f-step* [*of 0*] **have** *f* (*Suc 0*) = (*nrms*,*css*,*Normal s*)
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* **have** *inf* Γ *nrms css* (*Normal s*)
apply (*simp add*: *inf-def*)
apply (*rule-tac* $x{=}\lambda i.\ f\ (Suc\ i)$ **in** *exI*)
by *simp*
with *hyp* **show** *False* **..**
qed
next
case (*ExitBlockAbrupt abrs css s nrms*)
have *hyp*: ¬ *inf* Γ *abrs css* (*Normal s*) **by** *fact*

show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
assume *f0*: $f\ 0 = ([], (nrms,\ abrs)\ \#\ css,\ Abrupt\ s)$
with *f-step* [*of 0*] **have** $f\ (Suc\ 0) = (abrs,css,Normal\ s)$
by (*auto elim*: *step-Normal-elim-cases*)
with *f-step* **have** $inf\ \Gamma\ abrs\ css\ (Normal\ s)$
apply (*simp add*: *inf-def*)
apply (*rule-tac* $x=\lambda i.\ f\ (Suc\ i)$ **in** *exI*)
by *simp*
with *hyp* **show** *False* **..**
qed
next
case (*ExitBlockFault nrms css f abrs*)
show *?case*
by (*rule not-inf-Fault*)
next
case (*ExitBlockStuck nrms css abrs*)
show *?case*
by (*rule not-inf-Stuck*)
next
case (*Cons c s cs css*)
have *termi-c*: $\Gamma\vdash c \downarrow s$ **by** *fact*
have *hyp*: $\forall t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \longrightarrow \Gamma\vdash cs,css\Downarrow t \wedge \neg\ inf\ \Gamma\ cs\ css\ t$ **by** *fact*
show $\neg\ inf\ \Gamma\ (c\ \#\ cs)\ css\ s$
proof
assume $inf\ \Gamma\ (c\ \#\ cs)\ css\ s$
thus *False*
proof (*rule infE*)
assume $inf\ \Gamma\ [c]\ []\ s$
with *terminates-impl-not-inf* [*OF termi-c*]
show *False* **..**
next
fix *t*
assume $\Gamma\vdash\langle c,s\rangle \Rightarrow t$ $inf\ \Gamma\ cs\ css\ t$
with *hyp* **show** *False* **by** *simp*
qed
qed
qed

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 terminatess-impl-no-inf-chain:
 assumes terminatess: Γ⊢cs,css⇓s
 shows ¬(∃f. f 0 = (cs,css,s) ∧ (∀i::nat. Γ⊢f i →⁺ f(Suc i)))
proof –
  have wf({(y,x). Γ⊢(cs,css,s) →* x ∧ Γ⊢x → y}⁺)
  proof (rule wf-trancl)
    show wf {(y, x). Γ⊢(cs, css, s) →* x ∧ Γ⊢x → y}
    proof (simp only: wf-iff-no-infinite-down-chain,clarify,simp)
      fix f
      assume ∀i. Γ⊢(cs, css, s) →* f i ∧ Γ⊢f i → f (Suc i)
      hence ∃f. f 0 = (cs, css, s) ∧ (∀i. Γ⊢f i → f (Suc i))
        by (rule renumber [to-pred])
      moreover from terminatess
      have ¬ (∃f. f 0 = (cs, css, s) ∧ (∀i. Γ⊢f i → f (Suc i)))
        by (rule terminatess-impl-not-inf [unfolded inf-def])
      ultimately show False
        by simp
    qed
  qed
  hence ¬ (∃f. ∀i. (f (Suc i), f i)
                 ∈ {(y, x). Γ⊢(cs, css, s) →* x ∧ Γ⊢x → y}⁺)
    by (simp add: wf-iff-no-infinite-down-chain)
  thus ?thesis
  proof (rule contrapos-nn)
    assume ∃f. f 0 = (cs, css, s) ∧ (∀i. Γ⊢f i →⁺ f (Suc i))
    then obtain f where
      f0: f 0 = (cs, css, s) and
      seq: ∀i. Γ⊢f i →⁺ f (Suc i)
      by iprover
    show
      ∃f. ∀i. (f (Suc i), f i) ∈ {(y, x). Γ⊢(cs, css, s) →* x ∧ Γ⊢x → y}⁺
    proof (rule exI [where x=f],rule allI)
      fix i
      show (f (Suc i), f i) ∈ {(y, x). Γ⊢(cs, css, s) →* x ∧ Γ⊢x → y}⁺
      proof –
        {
          fix i have Γ⊢(cs,css,s) →* f i
          proof (induct i)
            case 0 show Γ⊢(cs, css, s) →* f 0
              by (simp add: f0)
          next
            case (Suc n)
            have Γ⊢(cs, css, s) →* f n  by fact
```

with *seq* **show** $\Gamma\vdash(cs,\ css,\ s) \rightarrow^* f\ (Suc\ n)$
by (*blast intro*: *tranclp-into-rtranclp rtranclp-trans*)
qed
}
hence $\Gamma\vdash(cs,css,s) \rightarrow^* f\ i$
by *iprover*
with *seq* **have**
$(f\ (Suc\ i),\ f\ i) \in \{(y,\ x).\ \Gamma\vdash(cs,\ css,\ s) \rightarrow^* x \wedge \Gamma\vdash x \rightarrow^+ y\}$
by *clarsimp*
moreover
have $\forall y.\ \Gamma\vdash f\ i \rightarrow^+ y \longrightarrow \Gamma\vdash(cs,\ css,\ s) \rightarrow^* f\ i \longrightarrow \Gamma\vdash(cs,\ css,\ s) \rightarrow^* y$
by (*blast intro*: *tranclp-into-rtranclp rtranclp-trans*)
ultimately
show *?thesis*
by (*subst lem*)
qed
qed
qed
qed

corollary *terminates-impl-no-inf-chain*:
$\Gamma\vdash c\downarrow s \Longrightarrow \neg(\exists f.\ f\ 0 = ([c],[],s) \wedge (\forall i{::}nat.\ \Gamma\vdash f\ i \rightarrow^+ f(Suc\ i)))$
by (*rule terminatess-impl-no-inf-chain*) (*iprover intro*: *terminatess.intros*)

definition
termi-call-steps :: $('s,'p,'f)\ body \Rightarrow (('s \times 'p) \times ('s \times 'p))set$
where
termi-call-steps Γ =
$\{((t,q),(s,p)).\ \Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ s\ \wedge$
$(\exists css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ s) \rightarrow^+ ([the\ (\Gamma\ q)],css,Normal\ t))\}$

Sequencing computations, or more exactly continuation stacks

primrec *seq*:: $(nat \Rightarrow 'a\ list) \Rightarrow nat \Rightarrow 'a\ list$
where
$seq\ css\ 0 = []\ |$
$seq\ css\ (Suc\ i) = css\ i@seq\ css\ i$

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 S
assume *inf*:
$\forall i.\ (\lambda(t,q)\ (s,p).\ \Gamma\vdash(the\ (\Gamma\ p)) \downarrow Normal\ s\ \wedge$
$(\exists css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ s) \rightarrow^+ ([the\ (\Gamma\ q)],css,Normal\ t)))$
$(S\ (Suc\ i))\ (S\ i)$
obtain $s\ p$ **where** $s = (\lambda i.\ fst\ (S\ i))$ **and** $p = (\lambda i.\ snd\ (S\ i))$
by *auto*

with *inf*
have *inf'*:
$\forall\, i.\ \Gamma\vdash(the\ (\Gamma\ (p\ i)))\downarrow Normal\ (s\ i) \wedge$
$(\exists\, css.\ \Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
$([the\ (\Gamma\ (p\ (Suc\ i)))],css,Normal\ (s\ (Suc\ i))))$
apply −
apply (*rule allI*)
apply (*erule-tac x=i* **in** *allE*)
apply *auto*
done
show *False*
proof −
from *inf'* — Skolemization of css with axiom of choice
have $\exists\, css.\ \forall\, i.\ \Gamma\vdash(the\ (\Gamma\ (p\ i)))\downarrow Normal\ (s\ i) \wedge$
$\Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
$([the\ (\Gamma\ (p\ (Suc\ i)))],css\ i,Normal\ (s\ (Suc\ i)))$
apply −
apply (*rule choice*)
by *blast*
then obtain *css* **where**
termi-css: $\forall\, i.\ \Gamma\vdash(the\ (\Gamma\ (p\ i)))\downarrow Normal\ (s\ i)$ **and**
step-css: $\forall\, i.\ \Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
$([the\ (\Gamma\ (p\ (Suc\ i)))],css\ i,Normal\ (s\ (Suc\ i)))$
by *blast*
define *f* **where** $f\ i = ([the\ (\Gamma\ (p\ i))],\ seq\ css\ i,Normal\ (s\ i)::('a,'c)\ xstate)$ **for**
i
have $f\ 0 = ([the\ (\Gamma\ (p\ 0))],[],Normal\ (s\ 0))$
by (*simp add*: *f-def*)
moreover
have $\forall\, i.\ \Gamma\vdash\ (f\ i) \rightarrow^{+}\ (f\ (i+1))$
proof
fix *i*
from *step-css* [*rule-format*, *of i*]
have $\Gamma\vdash([the\ (\Gamma\ (p\ i))],\ [],\ Normal\ (s\ i)) \rightarrow^{+}$
$([the\ (\Gamma\ (p\ (Suc\ i)))],\ css\ i,\ Normal\ (s\ (Suc\ i)))$.
from *app-css-steps* [*OF this*,*simplified*]
have $\Gamma\vdash([the\ (\Gamma\ (p\ i))],\ seq\ css\ i,\ Normal\ (s\ i)) \rightarrow^{+}$
$([the\ (\Gamma\ (p\ (Suc\ i)))],\ css\ i@seq\ css\ i,\ Normal\ (s\ (Suc\ i)))$.
thus $\Gamma\vdash\ (f\ i) \rightarrow^{+}\ (f\ (i+1))$
by (*simp add*: *f-def*)
qed
moreover from *termi-css* [*rule-format*, *of 0*]
have $\neg\ (\exists f.\ (f\ 0 = ([the\ (\Gamma\ (p\ 0))],[],Normal\ (s\ 0)) \wedge$
$(\forall\, i.\ \Gamma\vdash(f\ i) \rightarrow^{+} f(Suc\ i))))$
by (*rule terminates-impl-no-inf-chain*)
ultimately show *False*
by *auto*
qed
qed

An alternative proof using Hilbert-choice instead of axiom of choice.

theorem *wf* (*termi-call-steps* Γ)
proof (*simp only*: *termi-call-steps-def wf-iff-no-infinite-down-chain*,
    clarify,*simp*)
  fix *S*
  assume *inf*:
    $\forall\, i.\ (\lambda(t,q)\ (s,p).\ \Gamma\vdash(the\ (\Gamma\ p)) \downarrow Normal\ s\ \wedge$
      $(\exists\, css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ s) \rightarrow^{+} ([the\ (\Gamma\ q)],css,Normal\ t)))$
    $(S\ (Suc\ i))\ (S\ i)$
  obtain *s p* **where** $s = (\lambda i.\ fst\ (S\ i))$ **and** $p = (\lambda i.\ snd\ (S\ i))$
    by *auto*
  with *inf*
  have *inf'*:
    $\forall\, i.\ \Gamma\vdash(the\ (\Gamma\ (p\ i))) \downarrow Normal\ (s\ i)\ \wedge$
      $(\exists\, css.\ \Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
        $([the\ (\Gamma\ (p\ (Suc\ i)))],css,Normal\ (s\ (Suc\ i))))$
    apply –
    apply (*rule allI*)
    apply (*erule-tac x=i* **in** *allE*)
    apply *auto*
    done
  show *False*
  proof –
    define *CSS* **where** *CSS i* = (*SOME css*.
      $\Gamma\vdash([the\ (\Gamma\ (p\ i))],[],\ Normal\ (s\ i)) \rightarrow^{+}$
      $([the\ (\Gamma\ (p\ (i+1)))],css,Normal\ (s\ (i+1))))$ **for** *i*
    define *f* **where** $f\ i = ([the\ (\Gamma\ (p\ i))],\ seq\ CSS\ i,Normal\ (s\ i)::('a,'c)\ xstate)$
for *i*
    have $f\ 0 = ([the\ (\Gamma\ (p\ 0))],[],Normal\ (s\ 0))$
      by (*simp add*: *f-def*)
    moreover
    have $\forall\, i.\ \Gamma\vdash\ (f\ i) \rightarrow^{+} (f\ (i+1))$
    proof
      fix *i*
      from *inf'* [*rule-format*, *of i*] **obtain** *css* **where**
        css: $\Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
          $([the\ (\Gamma\ (p\ (i+1)))],css,Normal\ (s\ (i+1)))$
        by *fastforce*
      hence $\Gamma\vdash([the\ (\Gamma\ (p\ i))],\ seq\ CSS\ i,\ Normal\ (s\ i)) \rightarrow^{+}$
          $([the\ (\Gamma\ (p\ (i+1)))],\ CSS\ i\ @\ seq\ CSS\ i,\ Normal\ (s\ (i+1)))$
        apply –
        apply (*unfold CSS-def*)
        apply (*rule someI2*
          [**where** $P{=}\lambda css.$
            $\Gamma\vdash([the\ (\Gamma\ (p\ i))],[],Normal\ (s\ i)) \rightarrow^{+}$
              $([the\ (\Gamma\ (p\ (i+1)))],css,\ Normal\ (s\ (i+1)))$])
        apply (*rule css*)
        apply (*fastforce dest*: *app-css-steps*)
        done

thus $\Gamma\vdash (f\ i) \rightarrow^{+} (f\ (i{+}1))$
by (*simp add*: *f-def*)
qed
moreover from *inf′* [*rule-format*, *of 0*]
have $\Gamma\vdash$*the* $(\Gamma\ (p\ 0)) \downarrow$ *Normal* $(s\ 0)$
by *iprover*
then have $\neg\ (\exists f.\ (f\ 0 = ([the\ (\Gamma\ (p\ 0))],[],Normal\ (s\ 0))\ \wedge$
$(\forall i.\ \Gamma\vdash(f\ i) \rightarrow^{+} f(Suc\ i))))$
by (*rule terminates-impl-no-inf-chain*)
ultimately show *False*
by *auto*
qed
qed

lemma *not-inf-implies-wf*: **assumes** *not-inf*: $\neg$ *inf* Γ *cs css s*
shows *wf* $\{(c2,c1).\ \Gamma \vdash (cs,css,s) \rightarrow^{*} c1\ \wedge\ \Gamma \vdash c1 \rightarrow c2\}$
proof (*simp only*: *wf-iff-no-infinite-down-chain*,*clarify*, *simp*)
fix *f*
assume $\forall i.\ \Gamma\vdash(cs,\ css,\ s) \rightarrow^{*} f\ i\ \wedge\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
hence $\exists f.\ f\ 0 = (cs,\ css,\ s)\ \wedge\ (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i))$
by (*rule renumber* [*to-pred*])
moreover from *not-inf*
have $\neg\ (\exists f.\ f\ 0 = (cs,\ css,\ s)\ \wedge\ (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)))$
by (*unfold inf-def*)
ultimately show *False*
by *simp*
qed

lemma *wf-implies-termi-reach*:
assumes *wf*: *wf* $\{(c2,c1).\ \Gamma \vdash (cs,css,s) \rightarrow^{*} c1\ \wedge\ \Gamma \vdash c1 \rightarrow c2\}$
shows $\bigwedge cs1\ css1\ s1.\ [\![\Gamma \vdash (cs,css,s) \rightarrow^{*} c1;\ c1{=}(cs1,css1,s1)]\!] \Longrightarrow \Gamma\vdash cs1,css1 \Downarrow s1$
using *wf*
proof (*induct c1*, *simp*)
fix *cs1 css1 s1*
assume *reach*: $\Gamma\vdash(cs,\ css,\ s) \rightarrow^{*} (cs1,\ css1,\ s1)$
assume *hyp-raw*: $\bigwedge y\ cs2\ css2\ s2.\ [\![\Gamma \vdash (cs1,css1,s1) \rightarrow (cs2,css2,s2);$
$\Gamma \vdash (cs,css,s) \rightarrow^{*} (cs2,css2,s2);\ y{=}(cs2,css2,s2)]\!] \Longrightarrow$
$\Gamma\vdash cs2,css2 \Downarrow s2$
have *hyp*: $\bigwedge cs2\ css2\ s2.\ [\![\Gamma \vdash (cs1,css1,s1) \rightarrow (cs2,css2,s2)]\!] \Longrightarrow$
$\Gamma\vdash cs2,css2 \Downarrow s2$
apply –
apply (*rule hyp-raw*)
apply *assumption*
using *reach*
apply *simp*
apply (*rule refl*)
done
show $\Gamma\vdash cs1,css1 \Downarrow s1$
proof (*cases s1*)

case (*Normal s1'*)
show *?thesis*
proof (*cases cs1*)
  case *Nil*
  note *cs1-Nil* = *this*
  show *?thesis*
  proof (*cases css1*)
    case *Nil*
    with *cs1-Nil* **show** *?thesis*
      by (*auto intro*: *terminatess.intros*)
  next
    case (*Cons nrms-abrs css1'*)
    then obtain *nrms abrs* **where** *nrms-abrs*: *nrms-abrs*=(*nrms*,*abrs*)
      by (*cases nrms-abrs*)
    have $\Gamma \vdash$ ([],(*nrms*,*abrs*)#*css1'*,*Normal s1'*) $\rightarrow$ (*nrms*,*css1'*,*Normal s1'*)
      by (*rule step.intros*)
    from *hyp* [*simplified cs1-Nil Cons nrms-abrs Normal*, *OF this*]
    have $\Gamma\vdash$*nrms*,*css1'*$\Downarrow$*Normal s1'*.
    from *ExitBlockNormal* [*OF this*] *cs1-Nil Cons nrms-abrs Normal*
    show *?thesis*
      by *auto*
  qed
next
  case (*Cons c1 cs1'*)
  have $\Gamma\vdash$*c1*#*cs1'*,*css1*$\Downarrow$*Normal s1'*
  proof (*cases c1*)
    case *Skip*
    have $\Gamma \vdash$ (*Skip*#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$ (*cs1'*,*css1*,*Normal s1'*)
      by (*rule step.intros*)
    from *hyp* [*simplified Cons Skip Normal*, *OF this*]
    have $\Gamma\vdash$*cs1'*,*css1*$\Downarrow$*Normal s1'*.
    with *Normal Skip* **show** *?thesis*
      by (*auto intro*: *terminatess.intros terminates.intros*
            elim: *exec-Normal-elim-cases*)
  next
    case (*Basic f*)
    have $\Gamma \vdash$ (*Basic f*#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$ (*cs1'*,*css1*,*Normal* (*f s1'*))
      by (*rule step.intros*)
    from *hyp* [*simplified Cons Basic Normal*, *OF this*]
    have $\Gamma\vdash$*cs1'*,*css1*$\Downarrow$*Normal* (*f s1'*).
    with *Normal Basic* **show** *?thesis*
      by (*auto intro*: *terminatess.intros terminates.intros*
            elim: *exec-Normal-elim-cases*)
  next
    case (*Spec r*)
    with *Normal* **show** *?thesis*
      apply *simp*
      apply (*rule terminatess.Cons*)
      apply (*fastforce intro*: *terminates.intros*)

apply (*clarify*)
apply (*erule exec-Normal-elim-cases*)
apply *clarsimp*
apply (*rule hyp*)
apply (*fastforce intro*: *step.intros simp add*: *Cons Spec Normal*)
apply (*fastforce intro*: *terminatess-Stuck*)
done
next
case (*Seq* c_1 c_2)
have $\Gamma \vdash$ (*Seq* c_1 c_2#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$ (c_1#c_2#*cs1'*,*css1*,*Normal s1'*)
by (*rule step.intros*)
from *hyp* [*simplified Cons Seq Normal*, *OF this*]
have $\Gamma \vdash c_1$ # c_2 # *cs1'*,*css1*⇓*Normal s1'*.
with *Normal Seq* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)
next
case (*Cond b* c_1 c_2)
show *?thesis*
proof (*cases s1'* ∈ *b*)
case *True*
hence $\Gamma \vdash$(*Cond b* c_1 c_2#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$ (c_1#*cs1'*,*css1*,*Normal s1'*)
by (*rule step.intros*)
from *hyp* [*simplified Cons Cond Normal*, *OF this*]
have $\Gamma \vdash c_1$ # *cs1'*,*css1*⇓*Normal s1'*.
with *Normal Cond True* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)
next
case *False*
hence $\Gamma \vdash$(*Cond b* c_1 c_2#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$ (c_2#*cs1'*,*css1*,*Normal s1'*)
by (*rule step.intros*)
from *hyp* [*simplified Cons Cond Normal*, *OF this*]
have $\Gamma \vdash c_2$ # *cs1'*,*css1*⇓*Normal s1'*.
with *Normal Cond False* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)
qed
next
case (*While b c'*)
show *?thesis*
proof (*cases s1'* ∈ *b*)
case *True*
then have $\Gamma \vdash$(*While b c'* # *cs1'*, *css1*, *Normal s1'*) $\rightarrow$
(*c'* # *While b c'* # *cs1'*, *css1*, *Normal s1'*)
by (*rule step.intros*)

from *hyp* [*simplified Cons While Normal*, *OF this*]
have $\Gamma\vdash c'$ # *While b* c' # *cs1'*,*css1*$\Downarrow$*Normal s1'***.**
with *Cons While True Normal*
show *?thesis*
  by (*fastforce intro*: *terminatess.intros terminates.intros exec.intros*
    elim: *terminatess-elim-cases exec-Normal-elim-cases*)
next
case *False*
then
have $\Gamma\vdash$(*While b c'* # *cs1'*, *css1*, *Normal s1'*) $\rightarrow$ (*cs1'*, *css1*, *Normal s1'*)
  by (*rule step.intros*)
from *hyp* [*simplified Cons While Normal*, *OF this*]
have $\Gamma\vdash$*cs1'*,*css1*$\Downarrow$*Normal s1'***.**
with *Cons While False Normal*
show *?thesis*
  by (*fastforce intro*: *terminatess.intros terminates.intros exec.intros*
    elim: *terminatess-elim-cases exec-Normal-elim-cases*)
qed
next
case (*Call p*)
show *?thesis*
proof (*cases* Γ *p*)
case *None*
with *Call Normal* **show** *?thesis*
  by (*fastforce intro*: *terminatess.intros terminates.intros terminatess-Stuck*
    elim: *exec-Normal-elim-cases*)
next
case (*Some bdy*)
then
have $\Gamma\vdash$ (*Call p*#*cs1'*,*css1*,*Normal s1'*) $\rightarrow$
    ([*bdy*],(*cs1'*,*Throw*#*cs1'*)#*css1*,*Normal s1'*)
  by (*rule step.intros*)
from *hyp* [*simplified Cons Call Normal Some*, *OF this*]
have $\Gamma\vdash$[*bdy*],(*cs1'*, *Throw* # *cs1'*) # *css1*$\Downarrow$*Normal s1'***.**
with *Some Call Normal* **show** *?thesis*
  apply *simp*
  apply (*rule terminatess.intros*)
  apply (*blast elim*: *terminatess-elim-cases intro*: *terminates.intros*)
  apply *clarify*
  apply (*erule terminatess-elim-cases*)
  apply (*erule exec-Normal-elim-cases*)
  prefer *2*
  apply *simp*
  apply (*erule-tac x=t* **in** *allE*)
  apply (*case-tac t*)
  apply (*auto intro*: *terminatess-Stuck terminatess-Fault exec.intros*
    elim: *terminatess-elim-cases exec-Normal-elim-cases*)
  done
qed

next
case (*DynCom* c')
have $\Gamma \vdash$ (*DynCom* c'#*cs1*$'$,*css1*,*Normal s1*$'$) $\rightarrow$ (c' *s1*$'$#*cs1*$'$,*css1*,*Normal s1*$'$)
by (*rule step.intros*)
from *hyp* [*simplified Cons DynCom Normal*, *OF this*]
have $\Gamma\vdash c'$ *s1*$'$#*cs1*$'$,*css1*$\Downarrow$*Normal s1*$'$.
with *Normal DynCom* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros exec.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)
next
case (*Guard f g* c')
show *?thesis*
proof (*cases s1*$'$ $\in$ *g*)
case *True*
then have $\Gamma \vdash$ (*Guard f g* c'#*cs1*$'$,*css1*,*Normal s1*$'$) $\rightarrow$ (c'#*cs1*$'$,*css1*,*Normal s1*$'$)
by (*rule step.intros*)
from *hyp* [*simplified Cons Guard Normal*, *OF this*]
have $\Gamma\vdash c'$#*cs1*$'$,*css1*$\Downarrow$*Normal s1*$'$.
with *Normal Guard True* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros exec.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)
next
case *False*
with *Guard Normal* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminatess-Fault*
terminates.intros
elim: *exec-Normal-elim-cases*)
qed
next
case *Throw*
have $\Gamma \vdash$ (*Throw*#*cs1*$'$,*css1*,*Normal s1*$'$) $\rightarrow$ (*cs1*$'$,*css1*,*Abrupt s1*$'$)
by (*rule step.intros*)
from *hyp* [*simplified Cons Throw Normal*, *OF this*]
have $\Gamma\vdash$*cs1*$'$,*css1*$\Downarrow$*Abrupt s1*$'$.
with *Normal Throw* **show** *?thesis*
by (*auto intro*: *terminatess.intros terminates.intros*
elim: *exec-Normal-elim-cases*)
next
case (*Catch* c_1 c_2)
have $\Gamma \vdash$ (*Catch* c_1 c_2#*cs1*$'$,*css1*,*Normal s1*$'$) $\rightarrow$
([c_1], (*cs1*$'$,c_2#*cs1*$'$)# *css1*,*Normal s1*$'$)
by (*rule step.intros*)
from *hyp* [*simplified Cons Catch Normal*, *OF this*]
have $\Gamma\vdash$[c_1],(*cs1*$'$, c_2 # *cs1*$'$) # *css1*$\Downarrow$*Normal s1*$'$.
with *Normal Catch* **show** *?thesis*
by (*fastforce intro*: *terminatess.intros terminates.intros exec.intros*
elim: *terminatess-elim-cases exec-Normal-elim-cases*)

```
      qed
      with Cons Normal show ?thesis
        by simp
    qed
  next
    case (Abrupt s1')
    show ?thesis
    proof (cases cs1)
      case Nil
      note cs1-Nil = this
      show ?thesis
      proof (cases css1)
        case Nil
        with cs1-Nil show ?thesis by (auto intro: terminatess.intros)
      next
        case (Cons nrms-abrs css1')
        then obtain nrms abrs where nrms-abrs: nrms-abrs=(nrms,abrs)
          by (cases nrms-abrs)
        have Γ ⊢ ([],(nrms,abrs)#css1',Abrupt s1') → (abrs,css1',Normal s1')
          by (rule step.intros)
        from hyp [simplified cs1-Nil Cons nrms-abrs Abrupt, OF this]
        have Γ⊢abrs,css1'⇓Normal s1'.
        from ExitBlockAbrupt [OF this] cs1-Nil Cons nrms-abrs Abrupt
        show ?thesis
          by auto
      qed
    next
      case (Cons c1 cs1')
      have Γ⊢c1#cs1',css1⇓Abrupt s1'
      proof -
        have Γ ⊢ (c1#cs1',css1,Abrupt s1') → (cs1',css1,Abrupt s1')
          by (rule step.intros)
        from hyp [simplified Cons Abrupt, OF this]
        have Γ⊢cs1',css1⇓Abrupt s1'.
        with Cons Abrupt
        show ?thesis
          by (fastforce intro: terminatess.intros terminates.intros exec.intros
                    elim: terminatess-elim-cases exec-Normal-elim-cases)
      qed
      with Cons Abrupt show ?thesis by simp
    qed
  next
    case (Fault f)
    thus ?thesis by (auto intro: terminatess-Fault)
  next
    case Stuck
    thus ?thesis by (auto intro: terminatess-Stuck)
  qed
qed
```

lemma *not-inf-impl-terminatess*:
  assumes *not-inf*: $\neg$ *inf* Γ *cs css s*
  shows $\Gamma\vdash cs,css \Downarrow s$
proof −
  from *not-inf-implies-wf* [*OF not-inf*]
  have *wf*: *wf* $\{(c2,\ c1).\ \Gamma\vdash(cs,\ css,\ s) \rightarrow^* c1 \wedge \Gamma\vdash c1 \rightarrow c2\}$.
  show *?thesis*
    by (*rule wf-implies-termi-reach* [*OF wf*]) *auto*
qed

lemma *not-inf-impl-terminates*:
  assumes *not-inf*: $\neg$ *inf* Γ [*c*] [] *s*
  shows $\Gamma\vdash c\downarrow s$
proof −
  from *not-inf-impl-terminatess* [*OF not-inf*]
  have $\Gamma\vdash[c],[]\Downarrow s$.
  thus *?thesis*
    by (*auto elim*: *terminatess-elim-cases*)
qed

theorem *terminatess-iff-not-inf*:
  $\Gamma\vdash cs,css\Downarrow s = (\neg\ inf\ \Gamma\ cs\ css\ s)$
  apply *rule*
  apply (*erule terminatess-impl-not-inf*)
  apply (*erule not-inf-impl-terminatess*)
  done

corollary *terminates-iff-not-inf*:
  $\Gamma\vdash c\downarrow s = (\neg\ inf\ \Gamma\ [c]\ []\ s)$
  apply (*rule*)
  apply (*erule terminates-impl-not-inf*)
  apply (*erule not-inf-impl-terminates*)
  done

14.4 Completeness of Total Correctness Hoare Logic

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 *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 *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 $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma \vdash ([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c\#cs,css,Normal\ s))\}$
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 $\Gamma,\Theta \vdash_{t/F} \{s.\ s = Z \wedge \Gamma \vdash \langle Skip, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma \vdash ([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Skip\ \#\ cs,css,Normal\ s))\}$
$Skip$
$\{t.\ \Gamma \vdash \langle Skip, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Skip, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Skip* [*THEN conseqPre*]) (*blast intro*: *exec.Skip*)
next
case (*Basic f*)
show $\Gamma,\Theta \vdash_{t/F} \{s.\ s=Z \wedge \Gamma \vdash \langle Basic\ f, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma \vdash ([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Basic\ f\#cs,css,Normal\ s))\}$
$Basic\ f$
$\{t.\ \Gamma \vdash \langle Basic\ f, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma \vdash \langle Basic\ f, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule hoaret.Basic* [*THEN conseqPre*]) (*blast intro*: *exec.Basic*)
next
case (*Spec r*)
show $\Gamma,\Theta \vdash_{t/F} \{s.\ s=Z \wedge \Gamma \vdash \langle Spec\ r, Normal\ s\rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma \vdash ([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Spec\ r\#cs,css,Normal\ s))\}$
$Spec\ r$

$\{t.\ \Gamma\vdash\langle Spec\ r, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Spec\ r, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoaret.Spec* [*THEN conseqPre*])
apply (*clarsimp*)
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*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (c1\#cs,css,Normal\ s))\}$
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*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (c2\#cs,css,Normal\ s))\}$
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*
have *c1*: $\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (Seq\ c1\ c2\#cs,css,Normal$
$s))\}$
c1
$\{t.\ \Gamma\vdash\langle c1, Normal\ Z\rangle \Rightarrow Normal\ t \wedge$
$\Gamma\vdash\langle c2, Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (c2\#cs,css,Normal\ t))\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 *cs css*
assume $\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (Seq\ c1\ c2\ \#\ cs,css,Normal\ Z)$
thus $\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \to^* (c1\ \#\ cs,css,\ Normal\ Z)$
by (*blast intro*: *rtranclp-into-tranclp1* [*THEN tranclp-into-rtranclp*]
step.Seq)
next
fix *t*
assume $\Gamma\vdash\langle Seq\ c1\ c2, Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$

Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t*
thus Γ⊢⟨*c2*,*Normal t*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*))
by (*blast dest*: *Seq-NoFaultStuckD2*)
next
fix *cs css t*
assume Γ⊢([*the* (Γ *p*)],[],*Normal σ*) →* (*Seq c1 c2*#*cs*,*css*,*Normal Z*)
also have Γ⊢(*Seq c1 c2* # *cs*,*css*,*Normal Z*) → (*c1*#*c2*#*cs*,*css*,*Normal Z*)
by (*rule step.Seq*)
also assume Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t*
hence Γ⊢ (*c1*#*c2*#*cs*,*css*,*Normal Z*) →* (*c2*#*cs*,*css*,*Normal t*)
by (*rule exec-impl-steps*)
finally
show ∃ *cs css*. Γ⊢([*the* (Γ *p*)],[],*Normal σ*) →* (*c2* # *cs*,*css*, *Normal t*)
by *iprover*
next
fix *t*
assume Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Abrupt t*
thus Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*
by (*blast intro*: *exec.intros*)
qed
show Γ,Θ⊢$_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*Seq c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*))
∧
Γ⊢*the* (Γ *p*)↓*Normal σ* ∧
(∃ *cs css*. Γ⊢([*the* (Γ *p*)],[],*Normal σ*) →* (*Seq c1 c2*#*cs*,*css*,*Normal*
s))}
Seq c1 c2
{*t*. Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *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*:
∀ *Z*. Γ,Θ⊢$_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*c1*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*the* (Γ *p*)↓*Normal σ* ∧
(∃ *cs css*. Γ⊢([*the* (Γ *p*)],[], *Normal σ*) →* (*c1*#*cs*,*css*,*Normal s*))}
c1
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Abrupt t*}
using *Cond.hyps* **by** *iprover*
have
Γ,Θ⊢$_{t/F}$ ({*s*. *s*=*Z* ∧ Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*))
∧
Γ⊢*the* (Γ *p*)↓*Normal σ* ∧
(∃ *cs css*. Γ⊢([*the* (Γ *p*)],[],*Normal σ*) →* (*Cond b c1 c2*#*cs*,*css*,*Normal*
s))}
∩ *b*)
c1
{*t*. Γ⊢⟨*Cond b c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},

$\{t.\ \Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*safe*)
assume $Z \in b$ $\Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
thus $\Gamma\vdash\langle c1,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.CondTrue*)
next
fix *cs css*
assume $Z \in b$
$\Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\ \mathit{Normal}\ \sigma) \rightarrow^* (\mathit{Cond}\ b\ c1\ c2\ \#\ cs,css,\ \mathit{Normal}\ Z)$
thus $\exists\, cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\ \mathit{Normal}\ \sigma) \rightarrow^* (c1\ \#\ cs,css,\ \mathit{Normal}\ Z)$
by (*blast intro*: *rtranclp-into-tranclp1* [*THEN tranclp-into-rtranclp*]
step.CondTrue)
next
fix t **assume** $Z \in b$ $\Gamma\vdash\langle c1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
thus $\Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
by (*blast intro*: *exec.CondTrue*)
next
fix t **assume** $Z \in b$ $\Gamma\vdash\langle c1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
thus $\Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
by (*blast intro*: *exec.CondTrue*)
qed
moreover
have *hyp-c2*:
$\forall\, Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash\mathit{the}\ (\Gamma\ p)\downarrow\mathit{Normal}\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\mathit{Normal}\ \sigma) \rightarrow^* (c2\#cs,css,\mathit{Normal}\ s))\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Cond.hyps* **by** *iprover*
have
$\Gamma,\Theta\vdash_{t/F} (\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash\mathit{the}\ (\Gamma\ p)\downarrow\mathit{Normal}\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\mathit{Normal}\ \sigma) \rightarrow^*(\mathit{Cond}\ b\ c1\ c2\#cs,css,\ \mathit{Normal}$
$s))\}$
$\cap -b)$
$c2$
$\{t.\ \Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
assume $Z \notin b$ $\Gamma\vdash\langle \mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
thus $\Gamma\vdash\langle c2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.CondFalse*)
next
fix *cs css*
assume $Z \notin b$
$\Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\ \mathit{Normal}\ \sigma) \rightarrow^* (\mathit{Cond}\ b\ c1\ c2\ \#\ cs,css,\ \mathit{Normal}\ Z)$
thus $\exists\, cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\ \mathit{Normal}\ \sigma) \rightarrow^* (c2\ \#\ cs,css,\mathit{Normal}\ Z)$

by (*blast intro*: *rtranclp-into-tranclp1* [*THEN tranclp-into-rtranclp*] *step.CondFalse*)
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
$\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Cond\ b\ c1\ c2\#cs,css,Normal$
$s))\}$
$(Cond\ b\ c1\ c2)$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 $?P' = \lambda Z.\ \{t.\ (Z,t)\in ?unroll \wedge$
$(\forall\, e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*$
$(While\ b\ c\#cs,css,Normal\ t))\}$
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 $\Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*(While\ b\ c\#cs,css,Normal\ s))\}$
$(While\ b\ c)$
$\{t.\ \Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F} (?P'\ Z)\ (While\ b\ c)\ (?P'\ Z \cap -\ b),(?A\ Z)$
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix Z

from *While*
have *hyp-c*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c\ \#\ cs,css,Normal\ s))\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$ **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 $\tau\ s$
assume *asm*: $s{\in}\ \{\tau\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall\, e.\ (Z,e){\in}?unroll \longrightarrow e{\in}b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*$
$(While\ b\ c\#cs,css,Normal\ t))\}$
$\cap\ b$
then obtain *cs css* **where**
s-eq-τ: $s{=}\tau$ **and**
Z-s-unroll: $(Z,s) \in\ ?unroll$ **and**
noabort: $\forall\, e.\ (Z,e){\in}?unroll \longrightarrow e{\in}b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)$ **and**
termi: $\Gamma\vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma$ **and**
reach: $\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*$
$(While\ b\ c\#cs,css,Normal\ s)$**and**
s-in-b: $s{\in}b$
by *blast*
have *reach-c*:
$\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c\#While\ b\ c\#cs,css,Normal\ s)$
proof −
note *reach*
also from *s-in-b*
have $\Gamma\vdash(While\ b\ c\#cs,css,Normal\ s)\rightarrow$
$(c\#While\ b\ c\#cs,css,Normal\ s)$
by (*rule step.WhileTrue*)
finally
show *?thesis* **.**
qed
from *reach termi* **have**
termi-while: $\Gamma\vdash While\ b\ c \downarrow Normal\ s$
by (*rule steps-preserves-termination*)
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

```
              Γ⊢the (Γ p) ↓ Normal σ ∧
           (∃ cs css. Γ⊢([the (Γ p)],[],Normal σ) →* (c#cs,css,Normal t))} ∧
(∀ t. t ∈ {t. Γ⊢⟨c,Normal s⟩ ⇒ Normal t} ⟶
     t ∈ {t. (t,τ) ∈ ?r} ∩
         {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⊢the (Γ p) ↓ Normal σ ∧
            (∃ cs css. Γ⊢([the (Γ p)],[],Normal σ) →*
                        (While b c # cs,css,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 termi reach-c show ?C1
    by blast
next
  {
    fix t
    assume s-t: Γ⊢⟨c,Normal s⟩ ⇒ Normal t
    with s-eq-τ termi-while s-in-b have (t,τ) ∈ ?r
      by blast
    moreover
    from Z-s-unroll s-t s-in-b
    have (Z, t) ∈ ?unroll
      by (blast intro: rtrancl-into-rtrancl)
    moreover
    have Γ⊢([the (Γ p)],[],Normal σ) →* (While b c#cs,css,Normal t)
    proof −
      note reach-c
      also from s-t
      have Γ⊢(c#While b c#cs,css,Normal s)→*
              (While b c#cs,css, Normal t)
        by (rule exec-impl-steps)
      finally show ?thesis .
    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⊢the (Γ p) ↓ Normal σ ∧
            (∃ cs css. Γ⊢([the (Γ p)],[],Normal σ) →*
                        (While b c # cs,css, Normal t))
      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*: $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*$
$(While\ b\ c\#cs, css, Normal\ s))\}$
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*: $[\![e\in b;\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))]\!]$
$\Longrightarrow$ *?Prop r e*
assume *Z-r*:
$(Z,\ r) \in \{(Z,\ r).\ Z \in b \wedge \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ r\}$
with *WhileNoFault*
have $\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
from *hyp* [*OF e-in-b this*] **obtain**
cNoFault: $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$ **and**
Abrupt-r: $\forall\, u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow Abrupt\ u$
by *simp*

{
fix u **assume** $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u$
with *Abrupt-r* **have** $\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow Abrupt\ u$ **by** *simp*
moreover from *Z-r* **obtain**
$Z \in b$ $\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ r$
by *simp*
ultimately have $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u$
by (*blast intro*: *exec.intros*)
}
with *cNoFault* **show** *?Prop Z e*
by *iprover*
qed
}
with P **show** $s \in$ *?P′ s*
by *blast*
next
{
fix t
assume *termination*: $t \notin b$
assume $(Z,\ t) \in$ *?unroll*
hence $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t$
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
from *termination*
show $\Gamma\vdash\langle While\ b\ c,Normal\ t\rangle \Rightarrow Normal\ t$
by (*blast intro*: *exec.WhileFalse*)
next
fix Z r
assume *first-body*:
$(Z,\ r) \in \{(s,\ t).\ s \in b \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Normal\ t\}$
assume $(r,\ t) \in$ *?unroll*

assume *rest-loop*: $\Gamma\vdash\langle \mathit{While}\ b\ c,\ \mathit{Normal}\ r\rangle \Rightarrow \mathit{Normal}\ t$
show $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
proof $-$
from *first-body* **obtain**
$Z \in b\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ r$
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 $?P = \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q\ ,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{the}\ (\Gamma\ p)\downarrow \mathit{Normal}\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\mathit{Normal}\ \sigma) \rightarrow^*$
$(\mathit{Call}\ q\ \#\ cs,css,\mathit{Normal}\ s))\}$
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**
$\forall q\in \mathit{dom}\ \Gamma.\ \forall Z.$
$\Gamma,\Theta\vdash_{t/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{the}\ (\Gamma\ q),\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash(\mathit{the}\ (\Gamma\ q))\downarrow \mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{the}\ (\Gamma\ q),\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{the}\ (\Gamma\ q),\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*simp add*: *exec-Call-body′ noFaultStuck-Call-body′* [*simplified*]
terminates-Normal-Call-body)
from *Call-hyp q-defined* **have** *Call-hyp′*:
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ q\downarrow \mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$

$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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 *cs css*
assume
$\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*\ (Call\ q\ \#\ cs,css,Normal\ Z)$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma$
hence $\Gamma\vdash Call\ q\downarrow Normal\ Z$
by (*rule steps-preserves-termination*)
with *q-defined* **show** $\Gamma\vdash Call\ q\downarrow Normal\ Z$
by (*auto elim*: *terminates-Normal-elim-cases*)
next
fix *cs css*
assume *reach*:
$\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*\ (Call\ q\#cs,css,Normal\ Z)$
moreover have $\Gamma\vdash(Call\ q\ \#\ cs,css,\ Normal\ Z)\rightarrow$
$([the\ (\Gamma\ q)],(cs,Throw\#cs)\#css,\ Normal\ Z)$
by (*rule step.Call*) (*insert q-defined*,*auto*)
ultimately
have $\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^+\ ([the\ (\Gamma\ q)],(cs,Throw\#cs)\#css,Normal$
$Z)$
by (*rule rtranclp-into-tranclp1*)
moreover
assume *termi*: $\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma$
ultimately
show $((Z,q),\ \sigma,p)\in$ *termi-call-steps* Γ
by (*auto simp add*: *termi-call-steps-def*)
qed
qed
next
case (*DynCom c*)
have *hyp*:
$\bigwedge s'.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s = Z \wedge \Gamma\vdash\langle c\ s',Normal\ s\rangle \Rightarrow\notin(\{Stuck\}\cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*\ (c\ s'\#cs,css,Normal\ s))\}$
$(c\ s')$
$\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\}\cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*\ (DynCom\ c\#cs,css,Normal$

$s))\}$
 $(c\ Z)$
 $\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
 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 *cs css*
 assume $\Gamma\vdash([the\ (\Gamma\ p)],\ [],\ Normal\ \sigma) \rightarrow^*\ (DynCom\ c\ \#\ cs,\ css,\ Normal\ Z)$
 also have $\Gamma\vdash(DynCom\ c\ \#\ cs,\ css,\ Normal\ Z) \rightarrow (c\ Z\#cs,css,Normal\ Z)$
 by (*rule step.DynCom*)
 finally
 show $\exists\,cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],\ [],\ Normal\ \sigma) \rightarrow^*\ (c\ Z\ \#\ cs,\ css,\ Normal\ Z)$
 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*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$
 $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
 $\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
 $(\exists\,cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*\ (c\#cs,css,Normal\ s))\}$
 c
 $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
 $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
 using *Guard.hyps* **by** *iprover*
 show $\Gamma,\Theta\vdash_{t/F}\ \{s.\ s{=}Z \wedge \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
 $\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
 $(\exists\,cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^*\ (Guard\ f\ g\ c\#cs,css,Normal\ s))\}$
 $Guard\ f\ g\ c$
 $\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t\},$

$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

proof (*cases f* $\in$ *F*)

case *True*

have $\Gamma,\Theta\vdash_{t/F} (g \cap \{s.\ s=Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Guard\ f\ g\ c\#cs,css,Normal$
$s))\})$

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 *cs css*

assume $\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Guard\ f\ g\ c\#cs,css,Normal\ Z)$

also

assume $Z \in g$

hence $\Gamma\vdash(Guard\ f\ g\ c\#cs,css,Normal\ Z) \rightarrow (c\#cs,css,Normal\ Z)$

by (*rule step.Guard*)

finally show $\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c\#cs,css,Normal\ Z)$

by *iprover*

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 $\Gamma,\Theta\vdash_{t/F} (g \cap \{s.\ s=Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Guard\ f\ g\ c\#cs,css,Normal$
$s))\})$

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 *cs css*
  assume $\Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (Guard\ f\ g\ c\#cs,css,Normal\ Z)$
  also assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
  hence $Z \in g$
    using *False* **by** (*auto simp add*: *final-notin-def intro*: *exec.GuardFault*)
  hence $\Gamma\vdash(Guard\ f\ g\ c\#cs,css,Normal\ Z) \rightarrow (c\#cs,css,Normal\ Z)$
    by (*rule step.Guard*)
  finally show $\exists\ cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c\#cs,css,Normal\ Z)$
    by *iprover*
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 $\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
    $\Gamma\vdash the\ (\Gamma\ p)\downarrow Normal\ \sigma \wedge$
    $(\exists\ cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],\ Normal\ \sigma) \rightarrow^* (Throw\#cs,css,Normal\ s))\}$
    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*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s= Z \wedge \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c_1\#\ cs,\ css,Normal\ s))\}$
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*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s= Z \wedge \Gamma\vdash\langle c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c_2\#\ cs,\ css,Normal\ s))\}$
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
$\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma)\rightarrow^*(Catch\ c_1\ c_2\ \#cs,css,Normal$
$s))\}$
c_1
$\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t \wedge$
$\Gamma\vdash\langle c_2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault`(-F)) \wedge \Gamma\vdash the\ (\Gamma\ p) \downarrow Normal$
$\sigma \wedge$
$(\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],[],Normal\ \sigma) \rightarrow^* (c_2\#\ cs,\ css,Normal\ t))\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*clarify*,*safe*)
assume $\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
next
fix *cs css*
assume $\Gamma\vdash([the\ (\Gamma\ p)],\ [],\ Normal\ \sigma) \rightarrow^* (Catch\ c_1\ c_2\ \#\ cs,\ css,\ Normal\ Z)$
also have
$\Gamma\vdash(Catch\ c_1\ c_2\ \#\ cs,\ css,\ Normal\ Z) \rightarrow ([c_1],(cs,c_2\#cs)\#css,Normal\ Z)$
by (*rule step.Catch*)
finally
show $\exists\, cs\ css.\ \Gamma\vdash([the\ (\Gamma\ p)],\ [],\ Normal\ \sigma) \rightarrow^* (c_1\ \#\ cs,\ css,\ Normal\ Z)$
by *iprover*
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 *cs css t*
assume $\Gamma\vdash([\mathit{the}\ (\Gamma\ p)],\ [],\ \mathit{Normal}\ \sigma) \rightarrow^* (\mathit{Catch}\ c_1\ c_2\ \#\ cs,\ css,\ \mathit{Normal}\ Z)$
also have
$\Gamma\vdash(\mathit{Catch}\ c_1\ c_2\ \#\ cs,\ css,\ \mathit{Normal}\ Z) \rightarrow ([c_1],(cs,c_2\#cs)\#css,\mathit{Normal}\ Z)$
by (*rule step.Catch*)
also
assume $\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
hence $\Gamma\vdash([c_1],(cs,c_2\#cs)\#css,\mathit{Normal}\ Z) \rightarrow^* ([],(cs,c_2\#cs)\#css,\mathit{Abrupt}\ t)$
by (*rule exec-impl-steps*)
also
have $\Gamma\vdash([],(cs,c_2\#cs)\#css,\mathit{Abrupt}\ t) \rightarrow (c_2\#cs,css,\mathit{Normal}\ t)$
by (*rule step.intros*)
finally
show $\exists\ cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],\ [],\ \mathit{Normal}\ \sigma) \rightarrow^* (c_2\ \#\ cs,\ css,\ \mathit{Normal}\ t)$
by *iprover*
qed
moreover
have $\Gamma,\Theta\vdash_{t/F}$ $\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\ \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\ \wedge$
$\Gamma\vdash\mathit{the}\ (\Gamma\ p) \downarrow \mathit{Normal}\ \sigma\ \wedge$
$(\exists\ cs\ css.\ \Gamma\vdash([\mathit{the}\ (\Gamma\ p)],[],\mathit{Normal}\ \sigma) \rightarrow^* (c_2\#\ cs,\ css,\mathit{Normal}\ t))\}$
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* [*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
Call: $\forall\ q \in \mathit{dom}\ \Gamma.\ \forall\ Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\ \wedge$
$\Gamma\vdash \mathit{Call}\ q{\downarrow}\mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$(\{\sigma\} \cap \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{the}\ (\Gamma\ p),\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash\mathit{the}\ (\Gamma\ p){\downarrow}\mathit{Normal}\ s\})$
$\mathit{the}\ (\Gamma\ p)$
$\{t.\ \Gamma\vdash\langle \mathit{the}\ (\Gamma\ p),\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,

$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule conseqPre*)
apply (*rule Call-lemma′*)
apply (*rule Call*)
apply *blast*
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*)
apply *blast*
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*])
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 Γ,*Specs-wf p* σ
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ s\})$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
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

end

theory *Simpl-Heap*
imports *Main*
begin

14.5 References

definition *ref* = (*UNIV*::*nat set*)

typedef *ref* = *ref* **by** (*simp add*: *ref-def*)

code-datatype *Abs-ref*

lemma *finite-nat-ex-max*:
  assumes *fin*: *finite* (*N*::*nat set*)
  shows $\exists$ *m*. $\forall$ *n*$\in$*N*. *n* < *m*
using *fin*
proof (*induct*)
  case *empty*
  show *?case* **by** *auto*
next
  case (*insert k N*)
  have $\exists$ *m*. $\forall$ *n*$\in$*N*. *n* < *m* **by** *fact*
  then obtain *m* **where** *m-max*: $\forall$ *n*$\in$*N*. *n* < *m*..
  show $\exists$ *m*. $\forall$ *n*$\in$*insert k N*. *n* < *m*
  proof (*rule exI* [**where** *x*=*Suc* (*max k m*)])
  qed (*insert m-max*, *auto simp add*: *max-def*)
qed

lemma *infinite-nat*: ¬*finite* (*UNIV*::*nat set*)
proof
  assume *fin*: *finite* (*UNIV*::*nat set*)
  then obtain *m*::*nat* **where** $\forall$ *n*$\in$*UNIV*. *n* < *m*
    by (*rule finite-nat-ex-max* [*elim-format*]) *auto*
  moreover have *m*$\in$*UNIV*..
  ultimately show *False* **by** *blast*
qed

lemma *infinite-ref* [*simp*,*intro*]: ¬*finite* (*UNIV*::*ref set*)
proof
  assume *finite* (*UNIV*::*ref set*)
  hence *finite* (*range Rep-ref*)
    by *simp*
  moreover
  have *range Rep-ref* = *ref*
  proof
    show *range Rep-ref* $\subseteq$ *ref*
      by (*simp add*: *ref-def*)
  next
    show *ref* $\subseteq$ *range Rep-ref*
    proof
      fix *x*
      assume *x*: *x* $\in$ *ref*
      show *x* $\in$ *range Rep-ref*
        by (*rule Rep-ref-induct*) (*auto simp add*: *ref-def*)
    qed
  qed
  ultimately have *finite ref*

by *simp*
thus *False*
by (*simp add*: *ref-def infinite-nat*)
qed

consts *Null* :: *ref*

definition *new* :: *ref set* $\Rightarrow$ *ref* **where**
new A = (*SOME a. a* $\notin$ {*Null*} $\cup$ *A*)

Constant *Null* can be defined later on. Conceptually *Null* and *new* are *fixes* of a locale with *finite A* $\Longrightarrow$ *new A* $\notin$ *A* $\cup$ {*Null*}. But since definitions relative to a locale do not yet work in Isabelle2005 we use this workaround to avoid lots of parameters in definitions.

lemma *new-notin* [*simp*,*intro*]:
finite A $\Longrightarrow$ *new* (*A*) $\notin$ *A*
apply (*unfold new-def*)
apply (*rule someI2-ex*)
apply (*fastforce intro*: *ex-new-if-finite*)
apply *simp*
done

lemma *new-not-Null* [*simp*,*intro*]:
finite A $\Longrightarrow$ *new* (*A*) $\neq$ *Null*
apply (*unfold new-def*)
apply (*rule someI2-ex*)
apply (*fastforce intro*: *ex-new-if-finite*)
apply *simp*
done

end

15 Paths and Lists in the Heap

theory *HeapList*
imports *Simpl-Heap*
begin

Adapted from 'HOL/Hoare/Heap.thy'.

15.1 Paths in The Heap

primrec
Path :: *ref* $\Rightarrow$ (*ref* $\Rightarrow$ *ref*) $\Rightarrow$ *ref* $\Rightarrow$ *ref list* $\Rightarrow$ *bool*
where
Path x h y [] = (*x* = *y*) |
Path x h y (*p*#*ps*) = (*x* = *p* $\wedge$ *x* $\neq$ *Null* $\wedge$ *Path* (*h x*) *h y ps*)

lemma *Path-Null-iff* [*iff*]: $Path\ Null\ h\ y\ xs\ = (xs = [] \wedge y = Null)$
apply(*case-tac xs*)
apply *fastforce*
apply *fastforce*
done

lemma *Path-not-Null-iff* [*simp*]: $p{\neq}Null \Longrightarrow$
$Path\ p\ h\ q\ as = (as = [] \wedge q = p\ \vee\ (\exists\, ps.\ as = p\#ps \wedge Path\ (h\ p)\ h\ q\ ps\ ))$
apply(*case-tac as*)
apply *fastforce*
apply *fastforce*
done

lemma *Path-append* [*simp*]:
$\bigwedge p.\ Path\ p\ f\ q\ (as@bs)\ = (\exists\, y.\ Path\ p\ f\ y\ as \wedge Path\ y\ f\ q\ bs)$
by(*induct as*, *simp+*)

lemma *notin-Path-update*[*simp*]:
$\bigwedge p.\ u \notin set\ ps \Longrightarrow Path\ p\ (f(u := v))\ q\ ps\ = Path\ p\ f\ q\ ps$
by(*induct ps*, *simp*, *simp add*:*eq-sym-conv*)

lemma *Path-upd-same* [*simp*]:
$Path\ p\ (f(p{:=}p))\ q\ qs =$
$((p{=}Null \wedge q{=}Null \wedge qs = []) \vee (p{\neq}Null \wedge q{=}p \wedge (\forall\, x{\in}set\ qs.\ x{=}p)))$
by (*induct qs*) *auto*

Path-upd-same prevents $p \neq Null \Longrightarrow Path\ p\ (f(p := p))\ q\ qs = X$ from looping, because of *Path-not-Null-iff* and *fun-upd-apply*.

lemma *notin-Path-updateI* [*intro*]:
$[\![Path\ p\ h\ q\ ps\ ;\ r \notin set\ ps]\!] \Longrightarrow Path\ p\ (h(r := y))\ q\ ps$
by *simp*

lemma *Path-update-new* [*simp*]: $[\![set\ ps \subseteq set\ alloc]\!]$
$\Longrightarrow Path\ p\ (f(new\ (set\ alloc) := x))\ q\ ps\ = Path\ p\ f\ q\ ps$
by (*rule notin-Path-update*) *fastforce*

lemma *Null-notin-Path* [*simp*,*intro*]:
$\bigwedge p.\ Path\ p\ f\ q\ ps \Longrightarrow Null \notin set\ ps$
by(*induct ps*) *auto*

lemma *Path-snoc*:
$[\![Path\ p\ (f(a := q))\ a\ as\ ;\ a{\neq}Null]\!] \Longrightarrow Path\ p\ (f(a := q))\ q\ (as\ @\ [a])$
by *simp*

15.2 Lists on The Heap

15.2.1 Relational Abstraction

definition
$List :: ref \Rightarrow (ref \Rightarrow ref) \Rightarrow ref\ list \Rightarrow bool$ **where**

List p h ps = *Path p h Null ps*

lemma *List-empty* [*simp*]: *List p h* [] = (*p* = *Null*)
by(*simp add*:*List-def*)

lemma *List-cons* [*simp*]: *List p h* (*a*#*ps*) = (*p* = *a* ∧ *p*≠*Null* ∧ *List* (*h p*) *h ps*)
by(*simp add*:*List-def*)

lemma *List-Null* [*simp*]: *List Null h ps* = (*ps* = [])
by(*case-tac ps*, *simp-all*)

lemma *List-not-Null* [*simp*]: *p*≠*Null* ⟹
  List p h as = (∃ *ps*. *as* = *p*#*ps* ∧ *List* (*h p*) *h ps*)
by(*case-tac as*, *simp-all*, *fast*)

lemma *Null-notin-List* [*simp*,*intro*]: ⋀*p*. *List p h ps* ⟹ *Null* ∉ *set ps*
by (*simp add* : *List-def*)

theorem *notin-List-update*[*simp*]:
⋀*p*. *q* ∉ *set ps* ⟹ *List p* (*h*(*q* := *y*)) *ps* = *List p h ps*
apply(*induct ps*)
apply *simp*
apply *clarsimp*
done

lemma *List-upd-same-lemma*: ⋀*p*. *p* ≠ *Null* ⟹ ¬ *List p* (*h*(*p* := *p*)) *ps*
apply (*induct ps*)
apply *simp*
apply (*simp* (*no-asm-simp*) *del*: *fun-upd-apply*)
apply (*simp* (*no-asm-simp*) *only*: *fun-upd-apply refl if-True*)
apply *blast*
done

lemma *List-upd-same* [*simp*]: *List p* (*h*(*p*:=*p*)) *ps* = (*p* = *Null* ∧ *ps* = [])
apply (*cases p*=*Null*)
apply *simp*
apply (*fast dest*: *List-upd-same-lemma*)
done

List-upd-same prevents $p \neq Null \Longrightarrow List\ p\ (h(p := p))\ as = X$ from looping, because of *List-not-Null* and *fun-upd-apply*.

lemma *List-update-new* [*simp*]: ⟦*set ps* ⊆ *set alloc*⟧
  ⟹ *List p* (*h*(*new* (*set alloc*) := *x*)) *ps* = *List p h ps*
by (*rule notin-List-update*) *fastforce*

lemma *List-updateI* [*intro*]:
⟦*List p h ps*; *q* ∉ *set ps*⟧ ⟹ *List p* (*h*(*q* := *y*)) *ps*

by *simp*

lemma *List-unique*: $\bigwedge p\ bs.\ List\ p\ h\ as \Longrightarrow List\ p\ h\ bs \Longrightarrow as = bs$
by(*induct as*, *simp*, *clarsimp*)

lemma *List-unique1*: $List\ p\ h\ as \Longrightarrow \exists!as.\ List\ p\ h\ as$
by(*blast intro*:*List-unique*)

lemma *List-app*: $\bigwedge p.\ List\ p\ h\ (as@bs) = (\exists\, y.\ Path\ p\ h\ y\ as \wedge List\ y\ h\ bs)$
by(*induct as*, *simp*, *clarsimp*)

lemma *List-hd-not-in-tl*[*simp*]: $List\ (h\ p)\ h\ ps \Longrightarrow p \notin set\ ps$
apply (*clarsimp simp add*:*in-set-conv-decomp*)
apply(*frule List-app*[*THEN iffD1*])
apply(*fastforce dest*: *List-unique*)
done

lemma *List-distinct*[*simp*]: $\bigwedge p.\ List\ p\ h\ ps \Longrightarrow distinct\ ps$
apply(*induct ps*, *simp*)
apply(*fastforce dest*:*List-hd-not-in-tl*)
done

lemma *heap-eq-List-eq*:
  $\bigwedge p.\ \forall\, x \in set\ ps.\ h\ x = g\ x \Longrightarrow List\ p\ h\ ps = List\ p\ g\ ps$
  by (*induct ps*) *auto*

lemma *heap-eq-ListI*:
  assumes *list*: $List\ p\ h\ ps$
  assumes *hp-eq*: $\forall\, x \in set\ ps.\ h\ x = g\ x$
  shows $List\ p\ g\ ps$
  using *list*
  by (*simp add*: *heap-eq-List-eq* [*OF hp-eq*])

lemma *heap-eq-ListI1*:
  assumes *list*: $List\ p\ h\ ps$
  assumes *hp-eq*: $\forall\, x \in set\ ps.\ g\ x = h\ x$
  shows $List\ p\ g\ ps$
  using *list*
  by (*simp add*: *heap-eq-List-eq* [*OF hp-eq*])

The following lemmata are usefull for the simplifier to instantiate bound variables in the assumptions resp. conclusion, using the uniqueness of the List predicate

lemma *conj-impl-simp*: $(P \wedge Q \longrightarrow K) = (P \longrightarrow Q \longrightarrow K)$
by *auto*

lemma *List-unique-all-impl-simp* [*simp*]:

List p h ps ⟹ ($\forall$ *ps*. *List p h ps* ⟶ *P ps*) = *P ps*
by (*auto dest*: *List-unique*)

lemma *List-unique-ex-conj-simp* [*simp*]:
List p h ps ⟹ ($\exists$ *ps*. *List p h ps* ∧ *P ps*) = *P ps*
by (*auto dest*: *List-unique*)

15.3 Functional abstraction

definition
islist :: *ref* ⇒ (*ref* ⇒ *ref*) ⇒ *bool* **where**
islist p h = ($\exists$ *ps*. *List p h ps*)

definition
list :: *ref* ⇒ (*ref* ⇒ *ref*) ⇒ *ref list* **where**
list p h = (*THE ps*. *List p h ps*)

lemma *List-conv-islist-list*: *List p h ps* = (*islist p h* ∧ *ps* = *list p h*)
apply(*simp add*:*islist-def list-def*)
apply(*rule iffI*)
apply(*rule conjI*)
apply *blast*
apply(*subst the1-equality*)
apply(*erule List-unique1*)
apply *assumption*
apply(*rule refl*)
apply *simp*
apply (*clarify*)
apply (*rule theI*)
apply *assumption*
by (*rule List-unique*)

lemma *List-islist* [*intro*]:
List p h ps ⟹ *islist p h*
apply (*simp add*: *List-conv-islist-list*)
done

lemma *List-list*:
List p h ps ⟹ *list p h* = *ps*
apply (*simp only*: *List-conv-islist-list*)
done

lemma [*simp*]: *islist Null h*
by(*simp add*:*islist-def*)

lemma [*simp*]: *p*≠*Null* ⟹ *islist* (*h p*) *h* = *islist p h*

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

lemma [*simp*]: *list Null h* = []
by(*simp add*:*list-def*)

lemma *list-Ref-conv*[*simp*]:
$[\![$*islist* (*h p*) *h*; *p*≠*Null* $]\!]$ ⟹ *list p h* = *p* # *list* (*h p*) *h*
apply(*insert List-not-Null*[*of* - *h*])
apply(*fastforce simp*:*List-conv-islist-list*)
done

lemma [*simp*]: *islist* (*h p*) *h* ⟹ *p* ∉ *set*(*list* (*h p*) *h*)
apply(*insert List-hd-not-in-tl*[*of h*])
apply(*simp add*:*List-conv-islist-list*)
done

lemma *list-upd-conv*[*simp*]:
islist p h ⟹ *y* ∉ *set*(*list p h*) ⟹ *list p* (*h*(*y* := *q*)) = *list p h*
apply(*drule notin-List-update*[*of* - - *p h q*])
apply(*simp add*:*List-conv-islist-list*)
done

lemma *islist-upd*[*simp*]:
islist p h ⟹ *y* ∉ *set*(*list p h*) ⟹ *islist p* (*h*(*y* := *q*))
apply(*frule notin-List-update*[*of* - - *p h q*])
apply(*simp add*:*List-conv-islist-list*)
done

lemma *list-distinct*[*simp*]: *islist p h* ⟹ *distinct* (*list p h*)
apply (*clarsimp simp add*: *list-def islist-def*)
apply (*frule List-unique1*)
apply (*drule* (*1*) *the1-equality*)
apply *simp*
done

lemma *Null-notin-list* [*simp*,*intro*]: *islist p h* ⟹ *Null* ∉ *set* (*list p h*)
apply (*clarsimp simp add*: *list-def islist-def*)
apply (*frule List-unique1*)
apply (*drule* (*1*) *the1-equality*)
apply *simp*
done

end

theory *Generalise* **imports** *HOL−Statespace.DistinctTreeProver*
begin

lemma *protectRefl*: *PROP Pure.prop* (*PROP C*) $\Longrightarrow$ *PROP Pure.prop* (*PROP C*)
by (*simp add*: *prop-def*)

lemma *protectImp*:
assumes *i*: *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP Q*)
shows *PROP Pure.prop* (*PROP Pure.prop P* $\Longrightarrow$ *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

16 Facilitating the Hoare Logic

theory *Vcg*
imports *StateSpace HOL−Statespace.StateSpaceLocale Generalise*
keywords *procedures hoarestate* :: *thy-defn*
begin

axiomatization *NoBody*::('*s*,'*p*,'*f*) *com*

ML-file ⟨*hoare.ML*⟩

method-setup *hoare* = *Hoare.hoare*
  raw verification condition generator for Hoare Logic

method-setup *hoare-raw* = *Hoare.hoare-raw*
  even more raw verification condition generator for Hoare Logic

method-setup *vcg* = *Hoare.vcg*
  verification condition generator for Hoare Logic

method-setup *vcg-step* = *Hoare.vcg-step*
  single verification condition generation step with light simplification

method-setup *hoare-rule* = *Hoare.hoare-rule*
  apply single hoare rule and solve certain sideconditions

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 *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*

16.1 Some Fancy Syntax

reverse application

definition *rapp*:: '*a* ⇒ ('*a* ⇒ '*b*) ⇒ '*b* (**infixr** |> *60*)
  where *rapp x f* = *f x*

nonterminal
newinit **and**
newinits **and**
locinit **and**
locinits **and**
switchcase **and**
switchcases **and**
grds **and**
grd **and**
bdy **and**
basics **and**
basic **and**
basicblock

notation
Skip (*SKIP*) **and**
Throw (*THROW*)

syntax
-*raise*:: $'c \Rightarrow 'c \Rightarrow ('a,'b,'f)$ *com* ((*RAISE* - :==/ -) [*30*, *30*] *23*)
-*seq*::$('s,'p,'f)$ *com* $\Rightarrow ('s,'p,'f)$ *com* $\Rightarrow ('s,'p,'f)$ *com* (-;;/ - [*20*, *21*] *20*)
-*guarantee* :: $'s$ *set* $\Rightarrow$ *grd* (-$\surd$ [*1000*] *1000*)
-*guaranteeStrip*:: $'s$ *set* $\Rightarrow$ *grd* (-# [*1000*] *1000*)
-*grd* :: $'s$ *set* $\Rightarrow$ *grd* (- [*1000*] *1000*)
-*last-grd* :: *grd* $\Rightarrow$ *grds* (- *1000*)
-*grds* :: [*grd*, *grds*] $\Rightarrow$ *grds* (-,/ - [*999*,*1000*] *1000*)
-*guards* :: *grds* $\Rightarrow ('s,'p,'f)$ *com* $\Rightarrow ('s,'p,'f)$ *com*
((-/$\longmapsto$ -) [*60*, *21*] *23*)
-*quote* :: $'b$ => $('a$ => $'b)$
-*antiquoteCur0* :: $('a$ => $'b)$ => $'b$ (´- [*1000*] *1000*)
-*antiquoteCur* :: $('a$ => $'b)$ => $'b$
-*antiquoteOld0* :: $('a$ => $'b)$ => $'a$ => $'b$ (`- [*1000*,*1000*] *1000*)
-*antiquoteOld* :: $('a$ => $'b)$ => $'a$ => $'b$
-*Assert* :: $'a$ => $'a$ *set* (($\{\!|$-$|\!\}$) [*0*] *1000*)
-*AssertState* :: *idt* $\Rightarrow 'a$ => $'a$ *set* (($\{\!|$-. -$|\!\}$) [*1000*,*0*] *1000*)
-*Assign* :: $'b$ => $'b$ => $('a,'p,'f)$ *com* ((- :==/ -) [*30*, *30*] *23*)
-*Init* :: *ident* $\Rightarrow 'c \Rightarrow 'b \Rightarrow ('a,'p,'f)$ *com*
((´- :==$_-$/ -) [*30*,*1000*, *30*] *23*)
-*GuardedAssign*:: $'b$ => $'b$ => $('a,'p,'f)$ *com* ((- :==$_g$/ -) [*30*, *30*] *23*)
-*newinit* :: [*ident*,$'a$] $\Rightarrow$ *newinit* ((*2* ´- :==/ -))
:: *newinit* $\Rightarrow$ *newinits* (-)
-*newinits* :: [*newinit*, *newinits*] $\Rightarrow$ *newinits* (-,/ -)
-*New* :: [$'a$, $'b$, *newinits*] $\Rightarrow ('a,'b,'f)$ *com*
((- :==/(*2* *NEW* -/ [-])) [*30*, *65*, *0*] *23*)
-*GuardedNew* :: [$'a$, $'b$, *newinits*] $\Rightarrow ('a,'b,'f)$ *com*
((- :==$_g$/(*2* *NEW* -/ [-])) [*30*, *65*, *0*] *23*)
-*NNew* :: [$'a$, $'b$, *newinits*] $\Rightarrow ('a,'b,'f)$ *com*

((- :==/(2 NNEW -/ [-])) [30, 65, 0] 23)
-GuardedNNew :: ['a, 'b, newinits] ⇒ ('a,'b,'f) com
((- :==$_g$/(2 NNEW -/ [-])) [30, 65, 0] 23)

-Cond :: 'a bexp => ('a,'p,'f) com => ('a,'p,'f) com => ('a,'p,'f) com
((0IF (-)/ (2THEN/ -)/ (2ELSE -)/ FI) [0, 0, 0] 71)
-Cond-no-else:: 'a bexp => ('a,'p,'f) com => ('a,'p,'f) com
((0IF (-)/ (2THEN/ -)/ FI) [0, 0] 71)
-GuardedCond :: 'a bexp => ('a,'p,'f) com => ('a,'p,'f) com => ('a,'p,'f) com
((0IF$_g$ (-)/ (2THEN -)/ (2ELSE -)/ FI) [0, 0, 0] 71)
-GuardedCond-no-else:: 'a bexp => ('a,'p,'f) com => ('a,'p,'f) com
((0IF$_g$ (-)/ (2THEN -)/ FI) [0, 0] 71)
-While-inv-var :: 'a bexp => 'a assn ⇒ ('a × 'a) set ⇒ bdy
⇒ ('a,'p,'f) com
((0WHILE (-)/ INV (-)/ VAR (-) /-) [25, 0, 0, 81] 71)
-WhileFix-inv-var :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒
('z ⇒ ('a × 'a) set) ⇒ bdy
⇒ ('a,'p,'f) com
((0WHILE (-)/ FIX -./ INV (-)/ VAR (-) /-) [25, 0, 0, 0, 81] 71)
-WhileFix-inv :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒ bdy
⇒ ('a,'p,'f) com
((0WHILE (-)/ FIX -./ INV (-) /-) [25, 0, 0, 81] 71)
-GuardedWhileFix-inv-var :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒
('z ⇒ ('a × 'a) set) ⇒ bdy
⇒ ('a,'p,'f) com
((0WHILE$_g$ (-)/ FIX -./ INV (-)/ VAR (-) /-) [25, 0, 0, 0, 81] 71)
-GuardedWhileFix-inv-var-hook :: 'a bexp ⇒ ('z ⇒ 'a assn) ⇒
('z ⇒ ('a × 'a) set) ⇒ bdy
⇒ ('a,'p,'f) com
-GuardedWhileFix-inv :: 'a bexp => pttrn ⇒ ('z ⇒ 'a assn) ⇒ bdy
⇒ ('a,'p,'f) com
((0WHILE$_g$ (-)/ FIX -./ INV (-)/-) [25, 0, 0, 81] 71)

-GuardedWhile-inv-var::
'a bexp => 'a assn ⇒ ('a × 'a) set ⇒ bdy ⇒ ('a,'p,'f) com
((0WHILE$_g$ (-)/ INV (-)/ VAR (-) /-) [25, 0, 0, 81] 71)
-While-inv :: 'a bexp => 'a assn => bdy => ('a,'p,'f) com
((0WHILE (-)/ INV (-) /-) [25, 0, 81] 71)
-GuardedWhile-inv :: 'a bexp => 'a assn => ('a,'p,'f) com => ('a,'p,'f) com
((0WHILE$_g$ (-)/ INV (-) /-) [25, 0, 81] 71)
-While :: 'a bexp => bdy => ('a,'p,'f) com
((0WHILE (-) /-) [25, 81] 71)
-GuardedWhile :: 'a bexp => bdy => ('a,'p,'f) com
((0WHILE$_g$ (-) /-) [25, 81] 71)
-While-guard :: grds => 'a bexp => bdy => ('a,'p,'f) com
((0WHILE (-/⟼ (1-)) /-) [1000,25,81] 71)
-While-guard-inv:: grds ⇒'a bexp⇒'a assn⇒bdy ⇒ ('a,'p,'f) com
((0WHILE (-/⟼ (1-)) INV (-) /-) [1000,25,0,81] 71)
-While-guard-inv-var:: grds ⇒'a bexp⇒'a assn⇒('a×'a) set

⇒bdy ⇒ ('a,'p,'f) com
((0WHILE (-/⟼ (1-)) INV (-)/ VAR (-) /-) [1000,25,0,0,81] 71)
-WhileFix-guard-inv-var:: grds ⇒'a bexp⇒pttrn⇒('z⇒'a assn)⇒('z⇒('a×'a) set)
⇒bdy ⇒ ('a,'p,'f) com
((0WHILE (-/⟼ (1-)) FIX -./ INV (-)/ VAR (-) /-) [1000,25,0,0,0,81] 71)
-WhileFix-guard-inv:: grds ⇒'a bexp⇒pttrn⇒('z⇒'a assn)
⇒bdy ⇒ ('a,'p,'f) com
((0WHILE (-/⟼ (1-)) FIX -./ INV (-)/-) [1000,25,0,0,81] 71)

-Try-Catch:: ('a,'p,'f) com ⇒('a,'p,'f) com ⇒ ('a,'p,'f) com
((0TRY (-)/ (2CATCH -)/ END) [0,0] 71)

-DoPre :: ('a,'p,'f) com ⇒ ('a,'p,'f) com
-Do :: ('a,'p,'f) com ⇒ bdy ((2DO/ (-)) /OD [0] 1000)
-Lab:: 'a bexp ⇒ ('a,'p,'f) com ⇒ bdy
(-•/- [1000,71] 81)
:: bdy ⇒ ('a,'p,'f) com (-)
-Spec:: pttrn ⇒ 's set ⇒ ('s,'p,'f) com ⇒ 's set ⇒ 's set ⇒ ('s,'p,'f) com
((ANNO -. -/ (-)/ -,/-) [0,1000,20,1000,1000] 60)
-SpecNoAbrupt:: pttrn ⇒ 's set ⇒ ('s,'p,'f) com ⇒ 's set ⇒ ('s,'p,'f) com
((ANNO -. -/ (-)/ -) [0,1000,20,1000] 60)
-LemAnno:: 'n ⇒ ('s,'p,'f) com ⇒ ('s,'p,'f) com
((0 LEMMA (-)/ - END) [1000,0] 71)
-locnoinit :: ident ⇒ locinit (´-)
-locinit :: [ident,'a] ⇒ locinit ((2´- :==/ -))
:: locinit ⇒ locinits (-)
-locinits :: [locinit, locinits] ⇒ locinits (-,/ -)
-Loc:: [locinits,('s,'p,'f) com] ⇒ ('s,'p,'f) com
((2 LOC -;;/ (-) COL) [0,0] 71)
-Switch:: ('s ⇒ 'v) ⇒ switchcases ⇒ ('s,'p,'f) com
((0 SWITCH (-)/ - END) [22,0] 71)
-switchcase:: 'v set ⇒ ('s,'p,'f) com ⇒ switchcase (-⇒/ -)
-switchcasesSingle :: switchcase ⇒ switchcases (-)
-switchcasesCons:: switchcase ⇒ switchcases ⇒ switchcases
(-/ | -)
-Basic:: basicblock ⇒ ('s,'p,'f) com ((0BASIC/ (-)/ END) [22] 71)
-BasicBlock:: basics ⇒ basicblock (-)
-BAssign :: 'b => 'b => basic ((- :==/ -) [30, 30] 23)
:: basic ⇒ basics (-)
-basics :: [basic, basics] ⇒ basics (-,/ -)

syntax (*ASCII*)
-Assert :: 'a => 'a set (({|-|}) [0] 1000)
-AssertState :: idt ⇒ 'a ⇒ 'a set (({|-. -|}) [1000,0] 1000)
-While-guard :: grds => 'a bexp => bdy ⇒ ('a,'p,'f) com
((0WHILE (-|−> /-) /-) [0,0,1000] 71)
-While-guard-inv:: grds⇒'a bexp⇒'a assn⇒bdy ⇒ ('a,'p,'f) com

((0WHILE (-|-> /-) INV (-) /-) [0,0,0,1000] 71)
-guards :: grds ⇒ ('s,'p,'f) com ⇒ ('s,'p,'f) 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

{|s. P|} == {|-antiquoteCur((=) s) ∧ P |}
{|b|} => CONST Collect (-quote b)
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

-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 (λs. P) (λs. c) (λs. Q) (λs. A)
ANNO s. P c Q == ANNO s. P c Q,{}

-WhileFix-inv-var b z I V c => CONST whileAnnoFix {|b|} (λz. I) (λz. V) (λz. 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|} (λz. I) (λz. V) (λz. c)

-WhileFix-guard-inv-var gs b z I V c =>
CONST whileAnnoGFix gs {|b|} (λz. I) (λz. V) (λz. 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*)
(*-switchcasesSingle b*) => [*b*]
(*-switchcasesCons b bs*) => *CONST Cons b bs*
(*-Switch v vs*) => *CONST switch* (*-quote v*) *vs*

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
```
›

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
-faccess :: *'ref* ⇒ (*'ref* ⇒ *'v*) ⇒ *'v*
(-→- [*65*,*1000*] *100*)

syntax (*ASCII*)
-faccess :: *'ref* ⇒ (*'ref* ⇒ *'v*) ⇒ *'v*
(-->- [*65*,*1000*] *100*)

translations

p→*f* => *f p*
g→(*-antiquoteCur f*) <= *-antiquoteCur f g*

nonterminal *par* **and** *pars* **and** *actuals*

syntax
-par :: *'a* ⇒ *par* (-)
:: *par* ⇒ *pars* (-)
-pars :: [*par*,*pars*] ⇒ *pars* (-,/-)
-actuals :: *pars* ⇒ *actuals* ('(-'))
-actuals-empty :: *actuals* ('(''))

syntax *-Call* :: *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*) (*CALL* -- [*1000*,*1000*] *21*)

-GuardedCall :: *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*) (*CALL*$_g$ -- [*1000*,*1000*] *21*)
-CallAss:: *'a* ⇒ *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*)
(- :== *CALL* -- [*30*,*1000*,*1000*] *21*)
-Proc :: *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*) (*PROC* -- *21*)
-ProcAss:: *'a* ⇒ *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*)
(- :== *PROC* -- [*30*,*1000*,*1000*] *21*)
-GuardedCallAss:: *'a* ⇒ *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*)
(- :== *CALL*$_g$ -- [*30*,*1000*,*1000*] *21*)
-DynCall :: *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*) (*DYNCALL* -- [*1000*,*1000*] *21*)
-GuardedDynCall :: *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*) (*DYNCALL*$_g$ -- [*1000*,*1000*] *21*)
-DynCallAss:: *'a* ⇒ *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*)
(- :== *DYNCALL* -- [*30*,*1000*,*1000*] *21*)
-GuardedDynCallAss:: *'a* ⇒ *'p* ⇒ *actuals* ⇒ ((*'a*,*string*,*'f*) *com*)
(- :== *DYNCALL*$_g$ -- [*30*,*1000*,*1000*] *21*)

-Bind:: [*'s* ⇒ *'v*, *idt*, *'v* ⇒ (*'s*,*'p*,*'f*) *com*] ⇒ (*'s*,*'p*,*'f*) *com*
(- ≫ -./ - [*22*,*1000*,*21*] *21*)
-bseq::(*'s*,*'p*,*'f*) *com* ⇒ (*'s*,*'p*,*'f*) *com* ⇒ (*'s*,*'p*,*'f*) *com*
(-≫/ - [*22*, *21*] *21*)
-FCall :: [*'p*,*actuals*,*idt*,((*'a*,*string*,*'f*) *com*)]⇒ ((*'a*,*string*,*'f*) *com*)
(*CALL* -- ≫ -./ - [*1000*,*1000*,*1000*,*21*] *21*)

translations
-Bind e i c == *CONST bind* (*-quote e*) (λ*i*. *c*)
-FCall p acts i c == *-FCall p acts* (λ*i*. *c*)
-bseq c d == *CONST bseq c d*

nonterminal *modifyargs*

syntax
-may-modify :: [*'a*,*'a*,*modifyargs*] ⇒ *bool*
(- *may'-only'-modify'-globals* - *in* [-] [*100*,*100*,*0*] *100*)
-may-not-modify :: [*'a*,*'a*] ⇒ *bool*
(- *may'-not'-modify'-globals* - [*100*,*100*] *100*)
-may-modify-empty :: [*'a*,*'a*] ⇒ *bool*
(- *may'-only'-modify'-globals* - *in* [] [*100*,*100*] *100*)
-modifyargs :: [*id*,*modifyargs*] ⇒ *modifyargs* (-,/ -)
:: *id* => *modifyargs* (-)

translations
s may-only-modify-globals Z in [] => *s may-not-modify-globals Z*

definition *Let'*:: ['a, 'a => 'b] => 'b
  where *Let'* = *Let*

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.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.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.chopsfx Hoare.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
›
```

parse-translation ‹
```
 [(@{syntax-const -antiquoteCur},
   K (Hoare-Syntax.antiquote-varname-tr @{syntax-const -antiquoteCur}))]
›
```

parse-translation ‹
[(@{*syntax-const -antiquoteOld*}, *Hoare-Syntax.antiquoteOld-tr*),
(@{*syntax-const -Call*}, *Hoare-Syntax.call-tr false false*),
(@{*syntax-const -FCall*}, *Hoare-Syntax.fcall-tr*),
(@{*syntax-const -CallAss*}, *Hoare-Syntax.call-ass-tr false false*),
(@{*syntax-const -GuardedCall*}, *Hoare-Syntax.call-tr false true*),
(@{*syntax-const -GuardedCallAss*}, *Hoare-Syntax.call-ass-tr false true*),
(@{*syntax-const -Proc*}, *Hoare-Syntax.proc-tr*),
(@{*syntax-const -ProcAss*}, *Hoare-Syntax.proc-ass-tr*),
(@{*syntax-const -DynCall*}, *Hoare-Syntax.call-tr true false*),
(@{*syntax-const -DynCallAss*}, *Hoare-Syntax.call-ass-tr true false*),
(@{*syntax-const -GuardedDynCall*}, *Hoare-Syntax.call-tr true true*),
(@{*syntax-const -GuardedDynCallAss*}, *Hoare-Syntax.call-ass-tr true true*),
(@{*syntax-const -BasicBlock*}, *Hoare-Syntax.basic-assigns-tr*)]
›

parse-translation ‹
let
fun quote-tr ctxt [*t*] = *Hoare-Syntax.quote-tr ctxt* @{*syntax-const -antiquoteCur*}
t
| *quote-tr ctxt ts* = *raise TERM* (*quote-tr*, *ts*);
in [(@{*syntax-const -quote*}, *quote-tr*)] *end*
›

parse-translation ‹
[(@{*syntax-const -Assign*}, *Hoare-Syntax.assign-tr*),
(@{*syntax-const -raise*}, *Hoare-Syntax.raise-tr*),
(@{*syntax-const -New*}, *Hoare-Syntax.new-tr*),
(@{*syntax-const -NNew*}, *Hoare-Syntax.nnew-tr*),
(@{*syntax-const -GuardedAssign*}, *Hoare-Syntax.guarded-Assign-tr*),
(@{*syntax-const -GuardedNew*}, *Hoare-Syntax.guarded-New-tr*),
(@{*syntax-const -GuardedNNew*}, *Hoare-Syntax.guarded-NNew-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 -Basic*}, *Hoare-Syntax.basic-tr*)]
›

parse-translation ‹
[(@{*syntax-const -Init*}, *Hoare-Syntax.init-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*
›

syntax
-Measure:: (′*a* ⇒ *nat*) ⇒ (′*a* × ′*a*) *set*
(*MEASURE* - [*22*] *1*)
-Mlex:: (′*a* ⇒ *nat*) ⇒ (′*a* × ′*a*) *set* ⇒ (′*a* × ′*a*) *set*
(**infixr** <∗*MLEX*∗> *30*)

translations
MEASURE f => (*CONST measure*) (*-quote f*)
f <∗*MLEX*∗> *r* => (*-quote f*) <∗*mlex*∗> *r*

print-translation ‹
```
  let
    fun selector (Const (c,T)) = Hoare.is-state-var c
      | selector - = false;

    fun measure-tr′ ctxt ((t as (Abs (-,-,p)))::ts) =
          if Hoare-Syntax.antiquote-applied-only-to selector p
        then Hoare-Syntax.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.antiquote-applied-only-to selector p
         then Hoare-Syntax.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
```
›

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′)]
```
›

end

17 Examples using the Verification Environment

theory *VcgEx* **imports** ../*HeapList* ../*Vcg* **begin**

Some examples, especially the single-step Isar proofs are taken from `HOL/Isar_examples/HoareEx.thy`

17.1 State Spaces

First of all we provide a store of program variables that occur in the programs considered later. Slightly unexpected things may happen when attempting to work with undeclared variables.

record $'g\ vars = 'g\ state\ +$
  $A\text{-}'$ :: nat
  $I\text{-}'$ :: nat
  $M\text{-}'$ :: nat
  $N\text{-}'$ :: nat
  $R\text{-}'$ :: nat
  $S\text{-}'$ :: nat
  $B\text{-}'$ :: $bool$
  $Arr\text{-}'$ :: $nat\ list$
  $Abr\text{-}'$:: $string$

We decorate the state components in the record with the suffix -′, to avoid cluttering the namespace with the simple names that could no longer be used for logical variables otherwise.

We will first consider programs without procedures, later on we will regard procedures without global variables and finally we will get the full pictures: mutually recursive procedures with global variables (including heap).

17.2 Basic Examples

We look at few trivialities involving assignment and sequential composition, in order to get an idea of how to work with our formulation of Hoare Logic.

Using the basic rule directly is a bit cumbersome.

lemma $\Gamma\vdash \{|\acute{}N = 5|\}\ \acute{}N :== 2 * \acute{}N\ \{|\acute{}N = 10|\}$
  apply (*rule HoarePartial.Basic*) **apply** *simp*
  done

If we refer to components (variables) of the state-space of the program we always mark these with ´. It is the acute-symbol and is present on most keyboards. So all program variables are marked with the acute and all logical variables are not. The assertions of the Hoare tuple are ordinary Isabelle sets. As we usually want to refer to the state space in the assertions, we provide special brackets for them. They can be written as `{| |}` in ASCII or ⦃ ⦄ with symbols. Internally marking variables has two effects. First of all we refer to the implicit state and secondary we get rid of the suffix -′. So the assertion ⦃$\acute{}N = 5$⦄ internally gets expanded to $\{s.\ N\text{-}'\ s = 5\}$ written in ordinary set comprehension notation of Isabelle. It describes the set of states where the $N\text{-}'$ component is equal to 5.

Certainly we want the state modification already done, e.g. by simplification. The *vcg* method performs the basic state update for us; we may apply the Simplifier afterwards to achieve "obvious" consequences as well.

lemma $\Gamma\vdash \{\!|\mathit{True}|\!\}\ {}^\acute{}N :== \mathit{10}\ \{\!|{}^\acute{}N = \mathit{10}|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|\mathit{2} * {}^\acute{}N = \mathit{10}|\!\}\ {}^\acute{}N :== \mathit{2} * {}^\acute{}N\ \{\!|{}^\acute{}N = \mathit{10}|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|{}^\acute{}N = \mathit{5}|\!\}\ {}^\acute{}N :== \mathit{2} * {}^\acute{}N\ \{\!|{}^\acute{}N = \mathit{10}|\!\}$
  apply *vcg*
  apply *simp*
  done

lemma $\Gamma\vdash \{\!|{}^\acute{}N + \mathit{1} = a + \mathit{1}|\!\}\ {}^\acute{}N :== {}^\acute{}N + \mathit{1}\ \{\!|{}^\acute{}N = a + \mathit{1}|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|{}^\acute{}N = a|\!\}\ {}^\acute{}N :== {}^\acute{}N + \mathit{1}\ \{\!|{}^\acute{}N = a + \mathit{1}|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|a = a \wedge b = b|\!\}\ {}^\acute{}M :== a;;\ {}^\acute{}N :== b\ \{\!|{}^\acute{}M = a \wedge {}^\acute{}N = b|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|\mathit{True}|\!\}\ {}^\acute{}M :== a;;\ {}^\acute{}N :== b\ \{\!|{}^\acute{}M = a \wedge {}^\acute{}N = b|\!\}$
  by *vcg*

lemma $\Gamma\vdash \{\!|{}^\acute{}M = a \wedge {}^\acute{}N = b|\!\}$
  ${}^\acute{}I :== {}^\acute{}M;;\ {}^\acute{}M :== {}^\acute{}N;;\ {}^\acute{}N :== {}^\acute{}I$
  $\{\!|{}^\acute{}M = b \wedge {}^\acute{}N = a|\!\}$
  by *vcg*

We can also perform verification conditions generation step by step by using the *vcg-step* method.

lemma $\Gamma\vdash \{\!|{}^\acute{}M = a \wedge {}^\acute{}N = b|\!\}$
  ${}^\acute{}I :== {}^\acute{}M;;\ {}^\acute{}M :== {}^\acute{}N;;\ {}^\acute{}N :== {}^\acute{}I$
  $\{\!|{}^\acute{}M = b \wedge {}^\acute{}N = a|\!\}$
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  done

It is important to note that statements like the following one can only be proven for each individual program variable. Due to the extra-logical nature of record fields, we cannot formulate a theorem relating record selectors and updates schematically.

lemma $\Gamma\vdash$ $\{\!|\acute{}N = a|\!\}$ $\acute{}N$:== $\acute{}N$ $\{\!|\acute{}N = a|\!\}$
  by *vcg*

lemma $\Gamma\vdash\{s.\ x\text{-}'\ s = a\}$ $(Basic\ (\lambda s.\ x\text{-}'\text{-}update\ (x\text{-}'\ s)\ s))$ $\{s.\ x\text{-}'\ s = a\}$
  oops

In the following assignments we make use of the consequence rule in order to achieve the intended precondition. Certainly, the *vcg* method is able to handle this case, too.

lemma $\Gamma\vdash$ $\{\!|\acute{}M = \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
proof –
  have $\{\!|\acute{}M = \acute{}N|\!\} \subseteq \{\!|\acute{}M + 1 \neq \acute{}N|\!\}$
    by *auto*
  also have $\Gamma\vdash$ ... $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
    by *vcg*
  finally show *?thesis* **.**
qed

lemma $\Gamma\vdash$ $\{\!|\acute{}M = \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
proof –
  have $\bigwedge m\ n{::}nat.\ m = n \longrightarrow m + 1 \neq n$
      — inclusion of assertions expressed in "pure" logic,
      — without mentioning the state space
    by *simp*
  also have $\Gamma\vdash$ $\{\!|\acute{}M + 1 \neq \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
    by *vcg*
  finally show *?thesis* **.**
qed

lemma $\Gamma\vdash$ $\{\!|\acute{}M = \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
  apply *vcg*
  apply *simp*
  done

17.3 Multiplication by Addition

We now do some basic examples of actual `WHILE` programs. This one is a loop for calculating the product of two natural numbers, by iterated addition. We first give detailed structured proof based on single-step Hoare rules.

lemma $\Gamma\vdash$ $\{\!|\acute{}M = 0 \wedge \acute{}S = 0|\!\}$
      WHILE $\acute{}M \neq a$
      DO $\acute{}S$:== $\acute{}S + b$;; $\acute{}M$:== $\acute{}M + 1$ *OD*
      $\{\!|\acute{}S = a * b|\!\}$
proof –
  let $\Gamma\vdash$ - *?while* - = *?thesis*
  let $\{\!|\acute{}?inv|\!\} = \{\!|\acute{}S = \acute{}M * b|\!\}$

have $\{\!|\,{}´M = 0 \ \&\ ´S = 0|\!\} \subseteq \{\!|\,´?inv|\!\}$ **by** *auto*
also have $\Gamma\vdash$ ... *?while* $\{\!|\,´?inv \wedge \neg\ (´M \neq a)|\!\}$
proof
let $?c = ´S :== ´S + b;;\ ´M :== ´M + 1$
have $\{\!|\,´?inv \wedge ´M \neq a|\!\} \subseteq \{\!|\,´S + b = (´M + 1) * b|\!\}$
by *auto*
also have $\Gamma\vdash$ ... $?c\ \{\!|\,´?inv|\!\}$ **by** *vcg*
finally show $\Gamma\vdash \{\!|\,´?inv \wedge ´M \neq a|\!\}\ ?c\ \{\!|\,´?inv|\!\}$ **.**
qed
also have $\{\!|\,´?inv \wedge \neg\ (´M \neq a)|\!\} \subseteq \{\!|\,´S = a * b|\!\}$ **by** *auto*
finally show *?thesis* **by** *blast*
qed

The subsequent version of the proof applies the *vcg* method to reduce the Hoare statement to a purely logical problem that can be solved fully automatically. Note that we have to specify the `WHILE` loop invariant in the original statement.

lemma $\Gamma\vdash \{\!|\,´M = 0 \wedge ´S = 0|\!\}$
WHILE $´M \neq a$
INV $\{\!|\,´S = ´M * b|\!\}$
DO $´S :== ´S + b;;\ ´M :== ´M + 1$ *OD*
$\{\!|\,´S = a * b|\!\}$
apply *vcg*
apply *auto*
done

Here some examples of "breaking" out of a loop

lemma $\Gamma\vdash \{\!|\,´M = 0 \wedge ´S = 0|\!\}$
TRY
WHILE True
INV $\{\!|\,´S = ´M * b|\!\}$
DO IF $´M = a$ *THEN THROW ELSE* $´S :== ´S + b;;\ ´M :== ´M + 1$ *FI OD*
CATCH
SKIP
END
$\{\!|\,´S = a * b|\!\}$
apply *vcg*
apply *auto*
done

lemma $\Gamma\vdash \{\!|\,´M = 0 \wedge ´S = 0|\!\}$
TRY
WHILE True
INV $\{\!|\,´S = ´M * b|\!\}$
DO IF $´M = a$ *THEN* $´Abr :== ''Break'';;$*THROW*
ELSE $´S :== ´S + b;;\ ´M :== ´M + 1$
FI

```
      OD
    CATCH
      IF ´Abr = ''Break'' THEN SKIP ELSE Throw FI
    END
    {|´S = a * b|}
apply vcg
apply auto
done
```

Some more syntactic sugar, the label statement ... · ... as shorthand for the *TRY−CATCH* above, and the *RAISE* for an state-update followed by a *THROW*.

```
lemma Γ⊢ {|´M = 0 ∧ ´S = 0|}
    {|´Abr = ''Break''|}· WHILE True INV {|´S = ´M * b|}
     DO IF ´M = a THEN RAISE ´Abr :== ''Break''
        ELSE ´S :== ´S + b;; ´M :== ´M + 1
        FI
     OD
    {|´S = a * b|}
apply vcg
apply auto
done

lemma Γ⊢ {|´M = 0 ∧ ´S = 0|}
    TRY
      WHILE True
      INV {|´S = ´M * b|}
      DO IF ´M = a THEN RAISE ´Abr :== ''Break''
         ELSE ´S :== ´S + b;; ´M :== ´M + 1
         FI
      OD
    CATCH
      IF ´Abr = ''Break'' THEN SKIP ELSE Throw FI
    END
    {|´S = a * b|}
apply vcg
apply auto
done

lemma Γ⊢ {|´M = 0 ∧ ´S = 0|}
    {|´Abr = ''Break''|} · WHILE True
    INV {|´S = ´M * b|}
    DO IF ´M = a THEN RAISE ´Abr :== ''Break''
         ELSE ´S :== ´S + b;; ´M :== ´M + 1
         FI
    OD
    {|´S = a * b|}
apply vcg
apply auto
```

done

Blocks

lemma $\Gamma\vdash\{\!|\,{}^{\acute{}}I = i|\!\}$ *LOC* ´*I*;; ´*I* :== *2* *COL* $\{\!|\,{}^{\acute{}}I \le i|\!\}$
  apply *vcg*
  by *simp*
lemma $\Gamma\vdash$ $\{\!|\,{}^{\acute{}}N = n|\!\}$ *LOC* ´*N* :== *10*;; ´*N* :== ´*N* + *2* *COL* $\{\!|\,{}^{\acute{}}N = n|\!\}$
  by *vcg*

lemma $\Gamma\vdash$ $\{\!|\,{}^{\acute{}}N = n|\!\}$ *LOC* ´*N* :== *10*, ´*M*;; ´*N* :== ´*N* + *2* *COL* $\{\!|\,{}^{\acute{}}N = n|\!\}$
  by *vcg*

17.4 Summing Natural Numbers

We verify an imperative program to sum natural numbers up to a given limit. First some functional definition for proper specification of the problem.

primrec
  sum :: (*nat* => *nat*) => *nat* => *nat*
where
  sum f 0 = *0*
| *sum f* (*Suc n*) = *f n* + *sum f n*

syntax
  -sum :: *idt* => *nat* => *nat* => *nat*
    (*SUMM* -<-. - [*0*, *0*, *10*] *10*)
translations
  SUMM j<*k*. *b* == *CONST sum* (λ*j*. *b*) *k*

The following proof is quite explicit in the individual steps taken, with the *vcg* method only applied locally to take care of assignment and sequential composition. Note that we express intermediate proof obligation in pure logic, without referring to the state space.

theorem $\Gamma\vdash$ $\{\!|$*True*$|\!\}$
    ´*S* :== *0*;; ´*I* :== *1*;;
    WHILE ´*I* $\neq$ *n*
    DO
      ´*S* :== ´*S* + ´*I*;;
      ´*I* :== ´*I* + *1*
    OD
    $\{\!|\,{}^{\acute{}}S = (SUMM\ j{<}n.\ j)|\!\}$
  (**is** $\Gamma\vdash$ - (-;; *?while*) -)
proof −
  let *?sum* = λ*k*. *SUMM j*<*k*. *j*
  let *?inv* = λ*s i*. *s* = *?sum i*

  have $\Gamma\vdash$ $\{\!|$*True*$|\!\}$ ´*S* :== *0*;; ´*I* :== *1* $\{\!|$*?inv* ´*S* ´*I*$|\!\}$
  proof −
    have *True* $\longrightarrow$ *0* = *?sum 1*

```
    by simp
  also have Γ⊢ {|...|} ´S :== 0;; ´I :== 1 {|?inv ´S ´I|}
    by vcg
  finally show ?thesis .
 qed
 also have Γ⊢ {|?inv ´S ´I|} ?while {|?inv ´S ´I ∧ ¬ ´I ≠ n|}
 proof
  let ?body = ´S :== ´S + ´I;; ´I :== ´I + 1
  have ⋀s i. ?inv s i ∧ i ≠ n ⟶ ?inv (s + i) (i + 1)
    by simp
  also have Γ⊢ {|´S + ´I = ?sum (´I + 1)|} ?body {|?inv ´S ´I|}
    by vcg
  finally show Γ⊢ {|?inv ´S ´I ∧ ´I ≠ n|} ?body {|?inv ´S ´I|} .
 qed
 also have ⋀s i. s = ?sum i ∧ ¬ i ≠ n ⟶ s = ?sum n
  by simp
 finally show ?thesis .
qed
```

The next version uses the *vcg* method, while still explaining the resulting proof obligations in an abstract, structured manner.

```
theorem Γ⊢ {|True|}
          ´S :== 0;; ´I :== 1;;
          WHILE ´I ≠ n
          INV {|´S = (SUMM j<´I. j)|}
          DO
            ´S :== ´S + ´I;;
            ´I :== ´I + 1
          OD
         {|´S = (SUMM j<n. j)|}
proof −
 let ?sum = λk. SUMM j<k. j
 let ?inv = λs i. s = ?sum i

 show ?thesis
 proof vcg
  show ?inv 0 1 by simp
 next
  fix i s assume ?inv s i i ≠ n
  thus ?inv (s + i) (i + 1) by simp
 next
  fix i s assume x: ?inv s i ¬ i ≠ n
  thus s = ?sum n by simp
 qed
qed
```

Certainly, this proof may be done fully automatically as well, provided that the invariant is given beforehand.

```
theorem Γ⊢ {|True|}
```

´S :== 0;; ´I :== 1;;
WHILE ´I ≠ n
INV {|´S = (SUMM j<´I. j)|}
DO
´S :== ´S + ´I;;
´I :== ´I + 1
OD
{|´S = (SUMM j<n. j)|}
apply *vcg*
apply *auto*
done

17.5 SWITCH

lemma Γ⊢ {|´N = 5|} *SWITCH ´B*
{*True*} ⇒ ´N :== 7
| {*False*} ⇒ ´N :== 7
END
{|´N > 5|}
apply *vcg*
apply *simp*
done

lemma Γ⊢ {|´N = 5|} *SWITCH ´N*
{v. v < 5} ⇒ ´N :== 6
| {v. v ≥ 5} ⇒ ´N :== 7
END
{|´N > 5|}
apply *vcg*
apply *simp*
done

17.6 (Mutually) Recursive Procedures

17.6.1 Factorial

We want to define a procedure for the factorial. We first define a HOL functions that calculates it to specify the procedure later on.

primrec *fac*:: *nat* ⇒ *nat*
where
fac 0 = 1 |
fac (*Suc n*) = (*Suc n*) ∗ *fac n*

lemma *fac-simp* [*simp*]: *0 < i* ⟹ *fac i = i ∗ fac* (*i* − *1*)
by (*cases i*) *simp-all*

Now we define the procedure

procedures
Fac (*N*|*R*) = *IF ´N = 0 THEN ´R :== 1*

$ELSE$ $´R :== CALL\ Fac(´N - 1);;$
$´R :== ´N * ´R$
FI

A procedure is given by the signature of the procedure followed by the procedure body. The signature consists of the name of the procedure and a list of parameters. The parameters in front of the pipe | are value parameters and behind the pipe are the result parameters. Value parameters model call by value semantics. The value of a result parameter at the end of the procedure is passed back to the caller.

Behind the scenes the *procedures* command provides us convenient syntax for procedure calls, defines a constant for the procedure body (named *Fac-body*) and creates some locales. The purpose of locales is to set up logical contexts to support modular reasoning. A locale is named *Fac-impl* and extends the *hoare* locale with a theorem Γ $''Fac'' = Fac\text{-}body$ that simply states how the procedure is defined in the procedure context. Check out the locales. The purpose of the locales is to give us easy means to setup the context in which we will prove programs correct. In these locales the procedure context Γ is fixed. So always use this letter in procedure specifications. This is crucial, if we later on prove some tuples under the assumption of some procedure specifications.

thm *Fac-body.Fac-body-def*
print-locale *Fac-impl*

To see how a call is syntactically translated you can switch off the printing translation via the configuration option *hoare-use-call-tr'*

context *Fac-impl*
begin

$´M :== CALL\ Fac(´N)$ is internally:

declare [[*hoare-use-call-tr'* = *false*]]

$call\ (\lambda s.\ s(|N\text{-}' := N\text{-}'\ s|))\ Fac\text{-}'proc\ (\lambda s\ t.\ s(|globals := globals\ t|))\ (\lambda i\ t.\ ´M :== R\text{-}'\ t)$

term $CALL\ Fac(´N,´M)$
declare [[*hoare-use-call-tr'* = *true*]]
end

Now let us prove that *Fac* meets its specification.

Procedure specifications are ordinary Hoare tuples. We use the parameterless call for the specification; $´R :== PROC\ Fac(´N)$ is syntactic sugar for *Call ''Fac''*. This emphasises that the specification describes the internal behaviour of the procedure, whereas parameter passing corresponds to the procedure call.

lemma (**in** *Fac-impl*)
  shows $\forall n.\ \Gamma,\Theta\vdash\{\!|\ \acute{}N{=}n|\!\}\ \ \ PROC\ Fac(\acute{}N,\acute{}R)\ \{\!|\ \acute{}R = fac\ n|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *simp*
  done

Since the factorial was implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from general rule for mutually recursive procedures):

$[\![\forall Z.\ \Gamma,\Theta \cup (\bigcup_Z \{(P\ Z,\ p,\ Q\ Z,\ A\ Z)\})\vdash_{/F} (P\ Z)\ the\ (\Gamma\ p)\ (Q\ Z),(A\ Z);$
$p \in dom\ \Gamma]\!]$
$\Longrightarrow \forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ Call\ p\ (Q\ Z),(A\ Z)$

The verification condition generator will infer the specification out of the context when it encounters a recursive call of the factorial.

We can also step through verification condition generation. When the verification condition generator encounters a procedure call it tries to use the rule *ProcSpec*. To be successful there must be a specification of the procedure in the context.

lemma (**in** *Fac-impl*)
  shows $\forall n.\ \Gamma\vdash\{\!|\ \acute{}N{=}n|\!\}\ \ \acute{}R :== PROC\ Fac(\acute{}N)\ \{\!|\ \acute{}R = fac\ n|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *simp*
  done

Here some Isar style version of the proof

lemma (**in** *Fac-impl*)
  shows $\forall n.\ \Gamma\vdash\{\!|\ \acute{}N{=}n|\!\}\ \ \acute{}R :== PROC\ Fac(\acute{}N)\ \{\!|\ \acute{}R = fac\ n|\!\}$
proof (*hoare-rule HoarePartial.ProcRec1*)
  have *Fac-spec*: $\forall n.\ \Gamma,(\bigcup n.\ \{(\{\!|\ \acute{}N{=}n|\!\},\ Fac\text{-}'proc,\ \{\!|\ \acute{}R = fac\ n|\!\},\{\})\})$
    $\vdash \{\!|\ \acute{}N{=}n|\!\}\ \ \acute{}R :== PROC\ Fac(\acute{}N)\ \{\!|\ \acute{}R = fac\ n|\!\}$
    apply (*rule allI*)
    apply (*rule hoarep.Asm*)
    by *auto*
  show $\forall n.\ \Gamma,(\bigcup n.\ \{(\{\!|\ \acute{}N{=}n|\!\},\ Fac\text{-}'proc,\ \{\!|\ \acute{}R = fac\ n|\!\},\{\})\})$
    $\vdash \{\!|\ \acute{}N{=}n|\!\}\ IF\ \acute{}N = 0\ THEN\ \acute{}R :== 1$
    $ELSE\ \acute{}R :== CALL\ Fac(\acute{}N - 1);;\ \acute{}R :== \acute{}N * \acute{}R\ FI\ \{\!|\ \acute{}R = fac\ n|\!\}$
    apply *vcg*

apply *simp*
done
qed

To avoid retyping of potentially large pre and postconditions in the previous proof we can use the casual term abbreviations of the Isar language.

lemma (**in** *Fac-impl*)
shows $\forall n.\ \Gamma\vdash\{\!|\,\acute{}N{=}n|\!\}\ \acute{}R :== PROC\ Fac(\acute{}N)\ \{\!|\,\acute{}R = fac\ n|\!\}$
(**is** $\forall n.\ \Gamma\vdash(?Pre\ n)\ ?Fac\ (?Post\ n)$)
proof (*hoare-rule HoarePartial.ProcRec1*)
have *Fac-spec*: $\forall n.\ \Gamma,(\bigcup n.\ \{(?Pre\ n,\ Fac\text{-}'proc,\ ?Post\ n,\{\})\})$
$\vdash(?Pre\ n)\ ?Fac\ (?Post\ n)$
apply (*rule allI*)
apply (*rule hoarep.Asm*)
by *auto*
show $\forall n.\ \Gamma,(\bigcup n.\ \{(?Pre\ n,\ Fac\text{-}'proc,\ ?Post\ n,\{\})\})$
$\vdash\ (?Pre\ n)\ IF\ \acute{}N = 0\ THEN\ \acute{}R :== 1$
$ELSE\ \acute{}R :== CALL\ Fac(\acute{}N - 1);;\ \acute{}R :== \acute{}N * \acute{}R\ FI\ (?Post\ n)$
apply *vcg*
apply *simp*
done
qed

The previous proof pattern has still some kind of inconvenience. The augmented context is always printed in the proof state. That can mess up the state, especially if we have large specifications. This may be annoying if we want to develop single step or structured proofs. In this case it can be a good idea to introduce a new variable for the augmented context.

lemma (**in** *Fac-impl*) *Fac-spec*:
shows $\forall n.\ \Gamma\vdash\{\!|\,\acute{}N{=}n|\!\}\ \acute{}R :== PROC\ Fac(\acute{}N)\ \{\!|\,\acute{}R = fac\ n|\!\}$
(**is** $\forall n.\ \Gamma\vdash(?Pre\ n)\ ?Fac\ (?Post\ n)$)
proof (*hoare-rule HoarePartial.ProcRec1*)
define Θ' **where** $\Theta' = (\bigcup n.\ \{(?Pre\ n,\ Fac\text{-}'proc,\ ?Post\ n,\{\}{::}('a,\ 'b)\ vars\text{-}scheme\ set)\})$
have *Fac-spec*: $\forall n.\ \Gamma,\Theta'\vdash(?Pre\ n)\ ?Fac\ (?Post\ n)$
by (*unfold* Θ'*-def*, *rule allI*, *rule hoarep.Asm*) *auto*

We have to name the fact *Fac-spec*, so that the vcg can use the specification for the recursive call, since it cannot infer it from the opaque Θ'.

show $\forall \sigma.\ \Gamma,\Theta'\vdash\ (?Pre\ \sigma)\ IF\ \acute{}N = 0\ THEN\ \acute{}R :== 1$
$ELSE\ \acute{}R :== CALL\ Fac(\acute{}N - 1);;\ \acute{}R :== \acute{}N * \acute{}R\ FI\ (?Post\ \sigma)$
apply *vcg*
apply *simp*
done
qed

There are different rules available to prove procedure calls, depending on the kind of postcondition and whether or not the procedure is recursive or even mutually recursive. See for example *HoarePartial.ProcRec1*, *HoarePartial.ProcNoRec1*. They are all derived from the most general rule *HoarePartial.ProcRec*. All of them have some side-condition concerning definedness of the procedure. They can be solved in a uniform fashion. Thats why we have created the method *hoare-rule*, which behaves like the method *rule* but automatically tries to solve the side-conditions.

17.6.2 Odd and Even

Odd and even are defined mutually recursive here. In the *procedures* command we conjoin both definitions with *and*.

procedures
 odd(*N* | *A*) = *IF ´N=0 THEN ´A:==0*
    ELSE IF ´N=1 THEN CALL even (*´N* − *1*,*´A*)
     ELSE CALL odd (*´N* − *2*,*´A*)
     FI
    FI

and
 even(*N* | *A*) = *IF ´N=0 THEN ´A:==1*
    ELSE IF ´N=1 THEN CALL odd (*´N* − *1*,*´A*)
     ELSE CALL even (*´N* − *2*,*´A*)
     FI
    FI

print-theorems
thm *odd-body.odd-body-def*
thm *even-body.even-body-def*
print-locale *odd-even-clique*

To prove the procedure calls to *odd* respectively *even* correct we first derive a rule to justify that we can assume both specifications to verify the bodies. This rule can be derived from the general *HoarePartial.ProcRec* rule. An ML function does this work:

ML ‹*ML-Thms.bind-thm* (*ProcRec2*, *Hoare.gen-proc-rec* @{*context*} *Hoare.Partial* *2*)›

lemma (**in** *odd-even-clique*)
 shows *odd-spec*: ∀ *n*. Γ⊢⦃*´N=n*⦄ *´A* :== *PROC odd*(*´N*)
   ⦃(∃ *b*. *n* = *2* ∗ *b* + *´A*) ∧ *´A* < *2* ⦄ (**is** *?P1*)
 and *even-spec*: ∀ *n*. Γ⊢⦃*´N=n*⦄ *´A* :== *PROC even*(*´N*)
   ⦃(∃ *b*. *n* + *1* = *2* ∗ *b* + *´A*) ∧ *´A* < *2* ⦄ (**is** *?P2*)
proof −
 have *?P1* ∧ *?P2*
  apply (*hoare-rule ProcRec2*)

apply *vcg*
apply *clarsimp*
apply (*rule-tac x=b + 1* **in** *exI*)
apply *arith*
apply *vcg*
apply *clarsimp*
apply *arith*
done
thus *?P1 ?P2*
by *iprover+*
qed

17.7 Expressions With Side Effects

```
R := N++ + M++
```

lemma $\Gamma\vdash$ $\{\!|True|\!\}$
$\acute{}N \gg n.\ \acute{}N :== \acute{}N + 1 \gg$
$\acute{}M \gg m.\ \acute{}M :== \acute{}M + 1 \gg$
$\acute{}R :== n + m$
$\{\!|\acute{}R = \acute{}N + \acute{}M - 2|\!\}$
apply *vcg*
apply *simp*
done

```
R := Fac (N) + Fac (M)
```

lemma (**in** *Fac-impl*) **shows**
$\Gamma\vdash$ $\{\!|True|\!\}$
$CALL\ Fac(\acute{}N) \gg n.\ CALL\ Fac(\acute{}M) \gg m.$
$\acute{}R :== n + m$
$\{\!|\acute{}R = fac\ \acute{}N + fac\ \acute{}M|\!\}$
apply *vcg*
done

```
R := (Fac(Fac (N)))
```

lemma (**in** *Fac-impl*) **shows**
$\Gamma\vdash$ $\{\!|True|\!\}$
$CALL\ Fac(\acute{}N) \gg n.\ CALL\ Fac(n) \gg m.$
$\acute{}R :== m$
$\{\!|\acute{}R = fac\ (fac\ \acute{}N)|\!\}$
apply *vcg*
done

17.8 Global Variables and Heap

Now we define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

record *globals-list* =
  next-' :: *ref* ⇒ *ref*
  cont-' :: *ref* ⇒ *nat*

record *'g list-vars* = *'g state* +
  p-' :: *ref*
  q-' :: *ref*
  r-' :: *ref*
  root-' :: *ref*
  tmp-' :: *ref*

Updates to global components inside a procedure will always be propagated to the caller. This is implicitly done by the parameter passing syntax translations. The record containing the global variables must begin with the prefix "globals".

We first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter, and returns the result right into the first parameter.

procedures
  append(p,q|p) =
    IF ´p=Null THEN ´p :== ´q ELSE ´p →´next:== CALL append(´p→´next,´ q) FI

context *append-impl*
begin
declare [[*hoare-use-call-tr'* = *false*]]
term *CALL append(´p,´q,´p→´next)*
declare [[*hoare-use-call-tr'* = *true*]]
end

Below we give two specifications this time. One captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the other one is a pure frame condition. The list in the modifies clause has to list all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts of the lists will not be changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state t here.

The functional specification now introduces two logical variables besides the state space variable σ, namely *Ps* and *Qs*. They are universally quantified and range over both the pre and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax $\{|\sigma. \ldots|\}$

is a shorthand to fix the current state: $\{s.\ \sigma = s\ \ldots\}$.

lemma (**in** *append-impl*) *append-spec*:
  shows $\forall\sigma$ *Ps Qs*. $\Gamma\vdash$
      $\{\!|\sigma.$ *List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* $= \{\}|\!\}$
        ´p :== PROC append(´p,´q)
      $\{\!|$*List ´p ´next (Ps@Qs)* $\wedge$ $(\forall x.\ x \notin$ *set Ps* $\longrightarrow$ *´next* $x = {}^{\sigma}$*next* $x)|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *fastforce*
  done

The modifies clause is equal to a proper record update specification of the following form.

lemma $\{t.$ *t may-only-modify-globals Z in [next]*$\}$
    $=$
    $\{t.\ \exists$ *next. globals t=next-'-update* $(\lambda$-. *next*$)$ $($*globals Z*$)\}$
  apply (*unfold mex-def meq-def*)
  apply (*simp*)
  done

If the verification condition generator works on a procedure call it checks whether it can find a modified clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoarePartial.ProcSpec* is applied. Simplification of the procedure call means, that the "copy back" of the global components is simplified. Only those components that occur in the modifies clause will actually be copied back. This simplification is justified by the rule *HoarePartial.ProcModifyReturn*. So after this simplification all global components that do not appear in the modifies clause will be treated as local variables.

You can study the effect of the modifies clause on the following two examples, where we want to prove that (@) does not change the *cont* part of the heap.

lemma (**in** *append-impl*)
  shows $\Gamma\vdash$ $\{\!|$*´p=Null* $\wedge$ *´cont=c*$|\!\}$ *´p :== CALL append(´p,Null)* $\{\!|$*´cont=c*$|\!\}$
  apply *vcg*
  oops

To prove the frame condition, we have to tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies* It will also try to solve the verification conditions automatically.

lemma (**in** *append-impl*) *append-modifies*:
  shows
    $\forall\sigma.\ \Gamma\vdash\ \{\sigma\}$ *´p :== PROC append(´p,´q)*$\{t.$ *t may-only-modify-globals* σ *in* [*next*]$\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply (*vcg spec=modifies*)

done

lemma (**in** *append-impl*)
shows $\Gamma\vdash \{\!|\, ´p{=}Null \wedge ´cont{=}c |\!\}\ ´p{\rightarrow}´next :== CALL\ append(´p,Null)\ \{\!|\, ´cont{=}c |\!\}$
apply *vcg*
apply *simp*
done

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To verify the body of (@) we do not need the modifies clause, since the specification does not talk about *cont* at all, and we don't access *cont* inside the body. This may be different for more complex procedures.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but me may need the modifies clause to prove the functional specifications.

17.8.1 Insertion Sort

primrec $sorted :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$
where
$sorted\ le\ [] = True\ |$
$sorted\ le\ (x\#xs) = ((\forall y{\in}set\ xs.\ le\ x\ y) \wedge sorted\ le\ xs)$

procedures
insert(*r*,*p* | *p*) =
IF ´r=Null THEN SKIP
ELSE IF ´p=Null THEN ´p :== ´r;; ´p→´next :== Null
ELSE IF ´r→´cont ≤ ´p→´cont
THEN ´r→´next :== ´p;; ´p:==´r
ELSE ´p→´next :== CALL insert(´r,´p→´next)
FI
FI
FI

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state, even in the conclusion of the implication. The reason is, that we have separated out, that *cont* is not modified by the procedure, to the modifies clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitly state an equality in the postcondition. The reason is simple. If the postcondition would talk about *´cont* instead of ${}^{\sigma}cont$, we get a new instance of *cont* during verification and the postcondition would only state something about this new instance. But as the verification condition generator uses the modifies clause the caller of *insert* instead still has the old *cont* after the call. Thats the very reason for the modifies clause. So the caller and the specification will simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

lemma (**in** *insert-impl*) *insert-modifies*:
$\forall\sigma.\ \Gamma\vdash \{\sigma\}$ *´p :== PROC insert(´r,´p)*$\{t.$ *t may-only-modify-globals* σ *in* [*next*]$\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma (**in** *insert-impl*) *insert-spec*:

$$
\begin{array}{l}
\forall\sigma\ Ps\ .\ \Gamma\vdash \{\!|\sigma.\ List\ ´p\ ´next\ Ps \wedge sorted\ (\le)\ (map\ ´cont\ Ps) \wedge \\
\qquad\qquad ´r \ne Null \wedge ´r \notin set\ Ps|\!\} \\
\qquad ´p :== PROC\ insert(´r,´p) \\
\{\!|\exists Qs.\ List\ ´p\ ´next\ Qs \wedge sorted\ (\le)\ (map\ {}^{\sigma}cont\ \ Qs) \wedge \\
\qquad set\ Qs = insert\ {}^{\sigma}r\ (set\ Ps) \wedge \\
\qquad (\forall x.\ x \notin set\ Qs \longrightarrow ´next\ x = {}^{\sigma}next\ x)|\!\}
\end{array}
$$

apply (*hoare-rule HoarePartial.ProcRec1*)
apply *vcg*
apply (*intro conjI impI*)
apply *fastforce*
apply *fastforce*
apply *fastforce*
apply (*clarsimp*)
apply *force*
done

procedures
insertSort(*p* | *p*) =
  ´r:==Null;;
  WHILE (*´p* ≠ *Null*) *DO*
    ´q :== ´p;;
    ´p :== ´p→´next;;
    ´r :== CALL insert(´q,´r)
  OD;;
  ´p:==´r

lemma (**in** *insertSort-impl*) *insertSort-modifies*:
  shows
   $\forall \sigma.\ \Gamma\vdash \{\sigma\}$ *´p :== PROC insertSort(´p)*
      $\{t.$ *t may-only-modify-globals* σ *in* $[next]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

Insertion sort is not implemented recursively here but with a while loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. Therefore we will annotate the body during the proof with the rule *HoarePartial.annotateI*.

lemma (**in** *insertSort-impl*) *insertSort-body-spec*:
  shows $\forall \sigma\ Ps.\ \Gamma,\Theta\vdash \{|\sigma.$ *List ´p ´next Ps* $|\}$
      ´p :== PROC insertSort(´p)
    $\{|\exists\, Qs.$ *List ´p ´next Qs* $\wedge$ *sorted* $(\leq)$ (*map* $^{\sigma}$*cont Qs*) $\wedge$
     set Qs = set Ps$|\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply (*hoare-rule anno=*
    ´r :== Null;;
    WHILE ´p $\neq$ *Null*
    INV $\{|\exists\, Qs\ Rs.$ *List ´p ´next Qs* $\wedge$ *List ´r ´next Rs* $\wedge$
      set Qs $\cap$ *set Rs* $= \{\}$ $\wedge$
      sorted $(\leq)$ (*map ´cont Rs*) $\wedge$ *set Qs* $\cup$ *set Rs = set Ps* $\wedge$
      ´cont = $^{\sigma}$*cont* $|\}$
     DO ´q :== ´p;; ´p :== ´p$\rightarrow$*´next;; ´r :== CALL insert(´q,´r) OD;;*
    ´p :== ´r **in** *HoarePartial.annotateI*)
  apply *vcg*
  apply *fastforce*
  prefer *2*
  apply *fastforce*
  apply (*clarsimp*)
  apply (*rule-tac x=ps* **in** *exI*)
  apply (*intro conjI*)
  apply   (*rule heap-eq-ListI1*)
  apply   *assumption*
  apply   *clarsimp*
  apply   (*subgoal-tac x*$\neq$*p* $\wedge$ *x* $\notin$ *set Rs*)
  apply   *auto*
  done

17.8.2 Memory Allocation and Deallocation

The basic idea of memory management is to keep a list of allocated references in the state space. Allocation of a new reference adds a new reference to the list deallocation removes a reference. Moreover we keep a counter "free" for the free memory.

record *globals-list-alloc* = *globals-list* +
  alloc-′::ref list
  free-′::nat

record *′g list-vars′* = *′g list-vars* +
  i-′::nat
  first-′::ref

definition *sz* = (*2::nat*)

Restrict locale *hoare* to the required type.

locale *hoare-ex* =
  hoare Γ **for** Γ :: *′c* ⇀ ((*′a globals-list-alloc-scheme*, *′b*) *list-vars′-scheme*, *′c*, *′d*) *com*

lemma (**in** *hoare-ex*)
  Γ⊢ {|´*i* = *0* ∧ ´*first* = *Null* ∧ *n*∗*sz* ≤ ´*free*|}
    WHILE ´*i* < *n*
    INV {|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = ´*i* ∧ ´*i* ≤ *n* ∧
      set Ps ⊆ *set* ´*alloc* ∧ (*n* − ´*i*)∗*sz* ≤ ´*free*|}
    DO
      ´*p* :== *NEW sz* [´*cont*:==*0*,´*next*:== *Null*];;
      ´*p*→´*next* :== ´*first*;;
      ´*first* :== ´*p*;;
      ´*i* :== ´*i*+ *1*
    OD
    {|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = *n* ∧ *set Ps* ⊆ *set* ´*alloc*|}

apply (*vcg*)
apply *simp*
apply *clarsimp*
apply (*rule conjI*)
apply *clarsimp*
apply (*rule-tac x=new* (*set alloc*)#*Ps* **in** *exI*)
apply *clarsimp*
apply (*rule conjI*)
apply *fastforce*
apply (*simp add*: *sz-def*)
apply (*simp add*: *sz-def*)
apply *fastforce*
done

lemma (**in** *hoare-ex*)
  Γ⊢ {|´*i* = *0* ∧ ´*first* = *Null* ∧ *n*∗*sz* ≤ ´*free*|}
    WHILE ´*i* < *n*
    INV {|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = ´*i* ∧ ´*i* ≤ *n* ∧
      set Ps ⊆ *set* ´*alloc* ∧ (*n* − ´*i*)∗*sz* ≤ ´*free*|}

DO
  ´p :== NNEW sz [´cont:==0, ´next:== Null];;
  ´p→´next :== ´first;;
  ´first :== ´p;;
  ´i :== ´i+ 1
OD
$\{\!|\exists Ps.$ *List ´first ´next Ps* $\wedge$ *length Ps* $=$ *n* $\wedge$ *set Ps* $\subseteq$ *set ´alloc*$|\!\}$

apply (*vcg*)
apply *simp*
apply *clarsimp*
apply (*rule conjI*)
apply *clarsimp*
apply (*rule-tac x=new (set alloc)#Ps* **in** *exI*)
apply *clarsimp*
apply (*rule conjI*)
apply *fastforce*
apply (*simp add: sz-def*)
apply (*simp add: sz-def*)
apply *fastforce*
done

17.9 Fault Avoiding Semantics

If we want to ensure that no runtime errors occur we can insert guards into the code. We will not be able to prove any nontrivial Hoare triple about code with guards, if we cannot show that the guards will never fail. A trivial hoare triple is one with an empty precondition.

lemma $\Gamma\vdash$ $\{\!|$*True*$|\!\}$ $\{\!|$*´p≠Null*$|\!\}\longmapsto$ *´p→´next :== ´p* $\{\!|$*True*$|\!\}$
apply *vcg*
oops

lemma $\Gamma\vdash$ $\{\}$ $\{\!|$*´p≠Null*$|\!\}\longmapsto$ *´p→´next :== ´p* $\{\!|$*True*$|\!\}$
apply *vcg*
done

Let us consider this small program that reverts a list. At first without guards.

lemma (**in** *hoare-ex*) *rev-strip*:
  $\Gamma\vdash$ $\{\!|$*List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* $=$ $\{\}$ $\wedge$
    set Ps $\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
  WHILE ´p $\neq$ *Null*
  INV $\{\!|\exists ps\ qs.$ *List ´p ´next ps* $\wedge$ *List ´q ´next qs* $\wedge$ *set ps* $\cap$ *set qs* $=$ $\{\}$ $\wedge$
        rev ps @ qs = rev Ps @ Qs $\wedge$
        set ps $\subseteq$ *set ´alloc* $\wedge$ *set qs* $\subseteq$ *set ´alloc*$|\!\}$
  DO ´r :== ´p;;
    ´p :== ´p→ ´next;;
    ´r→´next :== ´q;;

$\quad \acute{}q :== \acute{}r\ \mathit{OD}$
$\{\!|\mathit{List}\ \acute{}q\ \acute{}\mathit{next}\ (\mathit{rev\ Ps}\ @\ \mathit{Qs}) \wedge \mathit{set\ Ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ Qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
apply (*vcg*)
apply *fastforce+*
done

If we want to ensure that we do not dereference *Null* or access unallocated memory, we have to add some guards.

locale *hoare-ex-guard* =
hoare Γ **for** $\Gamma :: {'c} \rightharpoonup (({'a}\ \mathit{globals\text{-}list\text{-}alloc\text{-}scheme}, {'b})\ \mathit{list\text{-}vars'\text{-}scheme}, {'c}, \mathit{bool})$ *com*

lemma
(**in** *hoare-ex-guard*)
$\Gamma\vdash \{\!|\mathit{List}\ \acute{}p\ \acute{}\mathit{next\ Ps} \wedge \mathit{List}\ \acute{}q\ \acute{}\mathit{next\ Qs} \wedge \mathit{set\ Ps} \cap \mathit{set\ Qs} = \{\} \wedge$
$\quad \mathit{set\ Ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ Qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
$\mathit{WHILE}\ \acute{}p \neq \mathit{Null}$
$\mathit{INV}\ \{\!|\exists\, \mathit{ps\ qs}.\ \mathit{List}\ \acute{}p\ \acute{}\mathit{next\ ps} \wedge \mathit{List}\ \acute{}q\ \acute{}\mathit{next\ qs} \wedge \mathit{set\ ps} \cap \mathit{set\ qs} = \{\} \wedge$
$\qquad \mathit{rev\ ps}\ @\ \mathit{qs} = \mathit{rev\ Ps}\ @\ \mathit{Qs} \wedge$
$\qquad \mathit{set\ ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
$\mathit{DO}\ \acute{}r :== \acute{}p;;$
$\quad \{\!|\acute{}p{\neq}\mathit{Null} \wedge \acute{}p{\in}\mathit{set}\ \acute{}\mathit{alloc}|\!\}{\longmapsto}\ \acute{}p :== \acute{}p{\rightarrow}\ \acute{}\mathit{next};;$
$\quad \{\!|\acute{}r{\neq}\mathit{Null} \wedge \acute{}r{\in}\mathit{set}\ \acute{}\mathit{alloc}|\!\}{\longmapsto}\ \acute{}r{\rightarrow}\acute{}\mathit{next} :== \acute{}q;;$
$\quad \acute{}q :== \acute{}r\ \mathit{OD}$
$\{\!|\mathit{List}\ \acute{}q\ \acute{}\mathit{next}\ (\mathit{rev\ Ps}\ @\ \mathit{Qs}) \wedge \mathit{set\ Ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ Qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
apply (*vcg*)
apply *fastforce+*
done

We can also just prove that no faults will occur, by giving the trivial postcondition.

lemma (**in** *hoare-ex-guard*) *rev-noFault*:
$\Gamma\vdash \{\!|\mathit{List}\ \acute{}p\ \acute{}\mathit{next\ Ps} \wedge \mathit{List}\ \acute{}q\ \acute{}\mathit{next\ Qs} \wedge \mathit{set\ Ps} \cap \mathit{set\ Qs} = \{\} \wedge$
$\quad \mathit{set\ Ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ Qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
$\mathit{WHILE}\ \acute{}p \neq \mathit{Null}$
$\mathit{INV}\ \{\!|\exists\, \mathit{ps\ qs}.\ \mathit{List}\ \acute{}p\ \acute{}\mathit{next\ ps} \wedge \mathit{List}\ \acute{}q\ \acute{}\mathit{next\ qs} \wedge \mathit{set\ ps} \cap \mathit{set\ qs} = \{\} \wedge$
$\qquad \mathit{rev\ ps}\ @\ \mathit{qs} = \mathit{rev\ Ps}\ @\ \mathit{Qs} \wedge$
$\qquad \mathit{set\ ps} \subseteq \mathit{set}\ \acute{}\mathit{alloc} \wedge \mathit{set\ qs} \subseteq \mathit{set}\ \acute{}\mathit{alloc}|\!\}$
$\mathit{DO}\ \acute{}r :== \acute{}p;;$
$\quad \{\!|\acute{}p{\neq}\mathit{Null} \wedge \acute{}p{\in}\mathit{set}\ \acute{}\mathit{alloc}|\!\}{\longmapsto}\ \acute{}p :== \acute{}p{\rightarrow}\ \acute{}\mathit{next};;$
$\quad \{\!|\acute{}r{\neq}\mathit{Null} \wedge \acute{}r{\in}\mathit{set}\ \acute{}\mathit{alloc}|\!\}{\longmapsto}\ \acute{}r{\rightarrow}\acute{}\mathit{next} :== \acute{}q;;$
$\quad \acute{}q :== \acute{}r\ \mathit{OD}$
$\mathit{UNIV}, \mathit{UNIV}$
apply (*vcg*)
apply *fastforce+*
done

lemma (**in** *hoare-ex-guard*) *rev-moduloGuards*:
$\Gamma\vdash_{/\{\mathit{True}\}} \{\!|\mathit{List}\ \acute{}p\ \acute{}\mathit{next\ Ps} \wedge \mathit{List}\ \acute{}q\ \acute{}\mathit{next\ Qs} \wedge \mathit{set\ Ps} \cap \mathit{set\ Qs} = \{\} \wedge$

$set\ Ps \subseteq set\ ´alloc \wedge set\ Qs \subseteq set\ ´alloc \}$
$WHILE\ ´p \neq Null$
$INV\ \{\exists ps\ qs.\ List\ ´p\ ´next\ ps \wedge List\ ´q\ ´next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$
$rev\ ps\ @\ qs = rev\ Ps\ @\ Qs \wedge$
$set\ ps \subseteq set\ ´alloc \wedge set\ qs \subseteq set\ ´alloc\}$
$DO\ ´r :== ´p;;$
$\{´p \neq Null \wedge ´p \in set\ ´alloc\} \surd \longmapsto ´p :== ´p \rightarrow ´next;;$
$\{´r \neq Null \wedge ´r \in set\ ´alloc\} \surd \longmapsto ´r \rightarrow ´next :== ´q;;$
$´q :== ´r\ OD$
$\{List\ ´q\ ´next\ (rev\ Ps\ @\ Qs) \wedge set\ Ps \subseteq set\ ´alloc \wedge set\ Qs \subseteq set\ ´alloc\}$
apply *vcg*
apply *fastforce+*
done

lemma *CombineStrip′*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ c'\ Q,A$
assumes *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}\ P\ c''\ UNIV,UNIV$
assumes c'': $c''=$ *mark-guards False* (*strip-guards* $(-F)$ c')
assumes c: $c =$ *mark-guards False* c'
shows $\Gamma,\Theta\vdash_{/\{\}}\ P\ c\ Q,A$
proof –
from *deriv-strip* [*simplified* c'']
have $\Gamma,\Theta\vdash\ P$ (*strip-guards* $(-\ F)$ c') $UNIV,UNIV$
by (*rule HoarePartialProps.MarkGuardsD*)
with *deriv*
have $\Gamma,\Theta\vdash\ P\ c'\ Q,A$
by (*rule HoarePartialProps.CombineStrip*)
hence $\Gamma,\Theta\vdash\ P$ *mark-guards False* $c'\ Q,A$
by (*rule HoarePartialProps.MarkGuardsI*)
thus *?thesis*
by (*simp add*: *c*)
qed

We can then combine the prove that no fault will occur with the functional proof of the programme without guards to get the full prove by the rule $[\![?\Gamma,?\Theta\vdash_{/?F}\ ?P\ ?c\ ?Q,?A;\ ?\Gamma,?\Theta\vdash\ ?P\ strip\text{-}guards\ (-\ ?F)\ ?c\ UNIV,UNIV]\!] \Longrightarrow\ ?\Gamma,?\Theta\vdash\ ?P\ ?c\ ?Q,?A$

lemma
(**in** *hoare-ex-guard*)
$\Gamma\vdash\ \{List\ ´p\ ´next\ Ps \wedge List\ ´q\ ´next\ Qs \wedge set\ Ps \cap set\ Qs = \{\} \wedge$
$set\ Ps \subseteq set\ ´alloc \wedge set\ Qs \subseteq set\ ´alloc\}$
$WHILE\ ´p \neq Null$
$INV\ \{\exists ps\ qs.\ List\ ´p\ ´next\ ps \wedge List\ ´q\ ´next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$
$rev\ ps\ @\ qs = rev\ Ps\ @\ Qs \wedge$
$set\ ps \subseteq set\ ´alloc \wedge set\ qs \subseteq set\ ´alloc\}$

DO ´r :== ´p;;
  {|´p≠Null ∧ ´p∈set ´alloc|}⟼ ´p :== ´p→ ´next;;
  {|´r≠Null ∧ ´r∈set ´alloc|}⟼ ´r→´next :== ´q;;
  ´q :== ´r OD
{|List ´q ´next (rev Ps @ Qs) ∧ set Ps ⊆ set ´alloc ∧ set Qs ⊆ set ´alloc|}

apply (*rule CombineStrip′* [*OF rev-moduloGuards rev-noFault*])
apply *simp*
apply *simp*
done

In the previous example the effort to split up the prove did not really pay off. But when we think of programs with a lot of guards and complicated specifications it may be better to first focus on a prove without the messy guards. Maybe it is possible to automate the no fault proofs so that it suffices to focus on the stripped program.

The purpose of guards is to watch for faults that can occur during evaluation of expressions. In the example before we watched for null pointer dereferencing or memory faults. We can also look for array index bounds or division by zero. As the condition of a while loop is evaluated in each iteration we cannot just add a guard before the while loop. Instead we need a special guard for the condition. Example: *WHILE* (*False*, *{|´p ≠ Null|}*)⟼ *´p→´next ≠ Null DO SKIP OD*

17.10 Circular Lists

definition
  distPath :: *ref* ⇒ (*ref* ⇒ *ref*) ⇒ *ref* ⇒ *ref list* ⇒ *bool* **where**
  distPath x next y as = (*Path x next y as* ∧ *distinct as*)

lemma *neq-dP*: ⟦*p* ≠ *q*; *Path p h q Ps*; *distinct Ps*⟧ ⟹
 ∃ *Qs*. *p*≠*Null* ∧ *Ps* = *p*#*Qs* ∧ *p* ∉ *set Qs*
by (*cases Ps*, *auto*)

lemma *circular-list-rev-I*:
  Γ⊢ *{|´root = r ∧ distPath ´root ´next ´root (r#Ps)|}*
   ´p :== ´root;; ´q :== ´root→´next;;
  WHILE ´q ≠ ´root
  INV {|∃ ps qs. distPath ´p ´next ´root ps ∧ distPath ´q ´next ´root qs ∧
      ´root = r ∧ r≠Null ∧ r ∉ set Ps ∧ set ps ∩ set qs = {} ∧
      Ps = (rev ps) @ qs |}
  DO ´tmp :== ´q;; ´q :== ´q→´next;; ´tmp→´next :== ´p;; ´p:==´tmp OD;;
  ´root→´next :== ´p
  {|´root = r ∧ distPath ´root ´next ´root (r#rev Ps)|}
apply (*simp only*:*distPath-def*)
apply *vcg*
apply (*rule-tac x*=[] **in** *exI*)

apply *fastforce*
apply *clarsimp*
apply (*drule* (*2*) *neq-dP*)
apply (*rule-tac* $x=q$ # *ps* **in** *exI*)
apply *clarsimp*
apply *fastforce*
done

lemma *path-is-list*: $\bigwedge a\ next\ b.\ [\![Path\ b\ next\ a\ Ps\ ;\ a \notin set\ Ps;\ a \neq Null]\!]$
$\Longrightarrow List\ b\ (next(a := Null))\ (Ps\ @\ [a])$
apply (*induct Ps*)
apply (*auto simp add*:*fun-upd-apply*)
done

The simple algorithm for acyclic list reversal, with modified annotations, works for cyclic lists as well.:

lemma *circular-list-rev-II*:
$\Gamma\vdash$
$\{\!|\acute{}p = r \wedge distPath\ \acute{}p\ \acute{}next\ \acute{}p\ (r\#Ps)|\!\}$
$\acute{}q{:}{=}{=}Null;;$
WHILE $\acute{}p \neq Null$
INV
$\{\!|\ ((\acute{}q = Null) \longrightarrow (\exists\,ps.\ distPath\ \acute{}p\ \acute{}next\ r\ ps\ \ \wedge\ \ ps = r\#Ps)) \wedge$
$((\acute{}q \neq Null) \longrightarrow (\exists\,ps\ qs.\ distPath\ \acute{}q\ \acute{}next\ r\ qs\ \ \wedge List\ \acute{}p\ \acute{}next\ ps\ \ \wedge$
$set\ ps \cap set\ qs = \{\} \wedge rev\ qs\ @\ ps = Ps@[r])) \wedge$
$\neg\ (\acute{}p = Null \wedge \acute{}q = Null \wedge r = Null\ )$
$|\!\}$
DO
$\acute{}tmp :{=}{=}\ \acute{}p;;\ \acute{}p :{=}{=}\ \acute{}p{\rightarrow}\acute{}next;;\ \acute{}tmp{\rightarrow}\acute{}next :{=}{=}\ \acute{}q;;\ \acute{}q{:}{=}{=}\acute{}tmp$
OD
$\{\!|\acute{}q = r \wedge distPath\ \acute{}q\ \acute{}next\ \acute{}q\ (r\ \#\ rev\ Ps)|\!\}$

apply (*simp only*:*distPath-def*)
apply *vcg*
apply *clarsimp*
apply *clarsimp*
apply (*case-tac* ($q = Null$))
apply (*fastforce intro*: *path-is-list*)
apply *clarify*
apply (*rule-tac* $x=psa$ **in** *exI*)
apply (*rule-tac* $x=\ p\ \#\ qs$ **in** *exI*)
apply *force*
apply *fastforce*
done

Although the above algorithm is more succinct, its invariant looks more involved. The reason for the case distinction on q is due to the fact that during execution, the pointer variables can point to either cyclic or acyclic structures.

When working on lists, its sometimes better to remove *fun-upd-apply* from the simpset, and instead include *fun-upd-same* and *fun-upd-other* to the simpset

lemma $\Gamma \vdash \{\sigma\}$
$\quad$ $\acute{I} :== \acute{M};;$
$\quad$ $ANNO\ \tau.\ \{\!|\tau.\ \acute{I} = {}^{\sigma}M|\!\}$
$\quad\quad$ $\acute{M} :== \acute{N};;\ \acute{N} :== \acute{I}$
$\quad\quad$ $\{\!|\acute{M} = {}^{\tau}N \wedge \acute{N} = {}^{\tau}I|\!\}$
$\quad$ $\{\!|\acute{M} = {}^{\sigma}N \wedge \acute{N} = {}^{\sigma}M|\!\}$
apply *vcg*
apply *auto*
done

lemma $\Gamma \vdash (\{\sigma\} \cap \{\!|\acute{M} = 0 \wedge \acute{S} = 0|\!\})$
$\quad$ $(ANNO\ \tau.\ (\{\tau\} \cap \{\!|\acute{A}={}^{\sigma}A \wedge \acute{I}={}^{\sigma}I \wedge \acute{M}=0 \wedge \acute{S}=0|\!\})$
$\quad$ $WHILE\ \acute{M} \neq \acute{A}$
$\quad$ $INV\ \{\!|\acute{S} = \acute{M} * \acute{I} \wedge \acute{A}={}^{\tau}A \wedge \acute{I}={}^{\tau}I|\!\}$
$\quad$ $DO\ \acute{S} :== \acute{S} + \acute{I};;\ \acute{M} :== \acute{M} + 1\ OD$
$\quad$ $\{\!|\acute{S} = {}^{\tau}A * {}^{\tau}I|\!\})$
$\quad$ $\{\!|\acute{S} = {}^{\sigma}A * {}^{\sigma}I|\!\}$
apply *vcg-step*
apply *vcg-step*
apply *simp*
apply *vcg-step*
apply *vcg-step*
apply *simp*
apply *vcg*
apply *simp*
apply *simp*
apply *vcg-step*
apply *auto*
done

Instead of annotations one can also directly use previously proven lemmas.

lemma *foo-lemma*: $\forall n\ m.\ \Gamma \vdash \{\!|\acute{N} = n \wedge \acute{M} = m|\!\}\ \acute{N} :== \acute{N} + 1;;\ \acute{M} :== \acute{M} + 1$
$\quad\quad$ $\{\!|\acute{N} = n + 1 \wedge \acute{M} = m + 1|\!\}$
by *vcg*

lemma $\Gamma \vdash \{\!|\acute{N} = n \wedge \acute{M} = m|\!\}\ LEMMA\ foo\text{-}lemma$
$\quad\quad$ $\acute{N} :== \acute{N} + 1;;\ \acute{M} :== \acute{M} + 1$
$\quad\quad$ $END;;$
$\quad\quad$ $\acute{N} :== \acute{N} + 1$
$\quad$ $\{\!|\acute{N} = n + 2 \wedge \acute{M} = m + 1|\!\}$

apply *vcg*
apply *simp*
done

lemma $\Gamma \vdash \{\!|\acute{}N = n \wedge \acute{}M = m|\!\}$
LEMMA foo-lemma
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1$
END;;
LEMMA foo-lemma
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1$
END
$\{\!|\acute{}N = n + 2 \wedge \acute{}M = m + 2|\!\}$
apply *vcg*
apply *simp*
done

lemma $\Gamma \vdash \{\!|\acute{}N = n \wedge \acute{}M = m|\!\}$
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1;;$
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1$
$\{\!|\acute{}N = n + 2 \wedge \acute{}M = m + 2|\!\}$
apply (*hoare-rule anno=*
LEMMA foo-lemma
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1$
END;;
LEMMA foo-lemma
$\acute{}N :== \acute{}N + 1;; \acute{}M :== \acute{}M + 1$
END
in *HoarePartial.annotate-normI*)
apply *vcg*
apply *simp*
done

Just some test on marked, guards

lemma $\Gamma \vdash \{\!|True|\!\}$ *WHILE* $\{\!|P\ \acute{}N\ |\!\}\surd, \{\!|Q\ \acute{}M|\!\}\#, \{\!|R\ \acute{}N|\!\} \longmapsto \acute{}N < \acute{}M$
INV $\{\!|\acute{}N < 2|\!\}$ *DO*
$\acute{}N :== \acute{}M$
OD
$\{\!|hard|\!\}$
apply *vcg*
oops

lemma $\Gamma \vdash_{/\{True\}} \{\!|True|\!\}$ *WHILE* $\{\!|P\ \acute{}N\ |\!\}\surd, \{\!|Q\ \acute{}M|\!\}\#, \{\!|R\ \acute{}N|\!\} \longmapsto \acute{}N < \acute{}M$
INV $\{\!|\acute{}N < 2|\!\}$ *DO*
$\acute{}N :== \acute{}M$
OD
$\{\!|hard|\!\}$
apply *vcg*
oops

term $\Gamma\vdash_{/\{True\}}$ $\{\!|True|\!\}$ *WHILE*$_g$ ´*N* < ´*Arr*!*i*
FIX Z.
INV $\{\!|$´*N* < *2*$|\!\}$

DO
´*N* :== ´*M*
OD
$\{\!|hard|\!\}$

lemma $\Gamma\vdash_{/\{True\}}$ $\{\!|True|\!\}$ *WHILE*$_g$ ´*N* < ´*Arr*!*i*
FIX Z.
INV $\{\!|$´*N* < *2*$|\!\}$
VAR arbitrary
DO
´*N* :== ´*M*
OD
$\{\!|hard|\!\}$
apply *vcg*
oops

lemma $\Gamma\vdash_{/\{True\}}$ $\{\!|True|\!\}$ *WHILE* $\{\!|P$ ´*N* $|\!\}\surd$, $\{\!|Q$ ´*M*$|\!\}\#$, $\{\!|R$ ´*N*$|\!\}\longmapsto$ ´*N* < ´*M*
FIX Z.
INV $\{\!|$´*N* < *2*$|\!\}$
VAR arbitrary
DO
´*N* :== ´*M*
OD
$\{\!|hard|\!\}$
apply *vcg*
oops

end

18 Examples using Statespaces

theory *VcgExSP* **imports** ../*HeapList* ../*Vcg* **begin**

18.1 State Spaces

First of all we provide a store of program variables that occur in the programs considered later. Slightly unexpected things may happen when attempting to work with undeclared variables.

hoarestate *state-space* =
A :: *nat*
I :: *nat*
M :: *nat*

N :: nat
R :: nat
S :: nat
B :: $bool$
Abr:: $string$

lemma (**in** *state-space*)$\Gamma\vdash$ $\{\!|\acute{}N = n|\!\}$ $LOC\ \acute{}N :== 10;;\ \acute{}N :== \acute{}N + 2\ COL$ $\{\!|\acute{}N = n|\!\}$
by *vcg*

Internally we decorate the state components in the statespace with the suffix -′, to avoid cluttering the namespace with the simple names that could no longer be used for logical variables otherwise.

We will first consider programs without procedures, later on we will regard procedures without global variables and finally we will get the full pictures: mutually recursive procedures with global variables (including heap).

18.2 Basic Examples

We look at few trivialities involving assignment and sequential composition, in order to get an idea of how to work with our formulation of Hoare Logic.

Using the basic rule directly is a bit cumbersome.

lemma (**in** *state-space*) $\Gamma\vdash$ $\{|\acute{}N = 5|\}$ $\acute{}N :== 2 * \acute{}N$ $\{|\acute{}N = 10|\}$
apply (*rule HoarePartial.Basic*)
apply *simp*
done

lemma (**in** *state-space*) $\Gamma\vdash$ $\{\!|True|\!\}$ $\acute{}N :== 10$ $\{\!|\acute{}N = 10|\!\}$
by *vcg*

lemma (**in** *state-space*) $\Gamma\vdash$ $\{\!|2 * \acute{}N = 10|\!\}$ $\acute{}N :== 2 * \acute{}N$ $\{\!|\acute{}N = 10|\!\}$
by *vcg*

lemma (**in** *state-space*) $\Gamma\vdash$ $\{\!|\acute{}N = 5|\!\}$ $\acute{}N :== 2 * \acute{}N$ $\{\!|\acute{}N = 10|\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *state-space*) $\Gamma\vdash$ $\{\!|\acute{}N + 1 = a + 1|\!\}$ $\acute{}N :== \acute{}N + 1$ $\{\!|\acute{}N = a + 1|\!\}$
by *vcg*

lemma (**in** *state-space*) $\Gamma\vdash$ $\{\!|\acute{}N = a|\!\}$ $\acute{}N :== \acute{}N + 1$ $\{\!|\acute{}N = a + 1|\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|a = a \wedge b = b|\!\}$ $\acute{}M$:== a;; $\acute{}N$:== b $\{\!|\acute{}M = a \wedge \acute{}N = b|\!\}$
  by *vcg*

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|\mathit{True}|\!\}$ $\acute{}M$:== a;; $\acute{}N$:== b $\{\!|\acute{}M = a \wedge \acute{}N = b|\!\}$
  by *vcg*

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|\acute{}M = a \wedge \acute{}N = b|\!\}$
  $\acute{}I$:== $\acute{}M$;; $\acute{}M$:== $\acute{}N$;; $\acute{}N$:== $\acute{}I$
  $\{\!|\acute{}M = b \wedge \acute{}N = a|\!\}$
  apply *vcg*
  apply *simp*
  done

We can also perform verification conditions generation step by step by using the *vcg-step* method.

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|\acute{}M = a \wedge \acute{}N = b|\!\}$
  $\acute{}I$:== $\acute{}M$;; $\acute{}M$:== $\acute{}N$;; $\acute{}N$:== $\acute{}I$
  $\{\!|\acute{}M = b \wedge \acute{}N = a|\!\}$
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *simp*
  done

In the following assignments we make use of the consequence rule in order to achieve the intended precondition. Certainly, the *vcg* method is able to handle this case, too.

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|\acute{}M = \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
proof −
  have $\{\!|\acute{}M = \acute{}N|\!\} \subseteq \{\!|\acute{}M + 1 \neq \acute{}N|\!\}$
    by *auto*
  also have $\Gamma\vdash \ldots$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
    by *vcg*
  finally show *?thesis* **.**
qed

lemma (**in** *state-space*)
  shows $\Gamma\vdash \{\!|\acute{}M = \acute{}N|\!\}$ $\acute{}M$:== $\acute{}M + 1$ $\{\!|\acute{}M \neq \acute{}N|\!\}$
proof −
  have $\bigwedge m\ n{::}nat.\ m = n \longrightarrow m + 1 \neq n$
    — inclusion of assertions expressed in "pure" logic,
    — without mentioning the state space

```
    by simp
  also have Γ⊢ ⦃´M + 1 ≠ ´N⦄ ´M :== ´M + 1 ⦃´M ≠ ´N⦄
    by vcg
  finally show ?thesis .
qed

lemma (in state-space)
  shows Γ⊢ ⦃´M = ´N⦄ ´M :== ´M + 1 ⦃´M ≠ ´N⦄
  apply vcg
  apply simp
  done
```

18.3 Multiplication by Addition

We now do some basic examples of actual `WHILE` programs. This one is a loop for calculating the product of two natural numbers, by iterated addition. We first give detailed structured proof based on single-step Hoare rules.

```
lemma (in state-space)
  shows Γ⊢ ⦃´M = 0 ∧ ´S = 0⦄
      WHILE ´M ≠ a
      DO ´S :== ´S + b;; ´M :== ´M + 1 OD
      ⦃´S = a ∗ b⦄
proof −
  let Γ⊢ - ?while - = ?thesis
  let ⦃´?inv⦄ = ⦃´S = ´M ∗ b⦄

  have ⦃´M = 0 & ´S = 0⦄ ⊆ ⦃´?inv⦄ by auto
  also have Γ⊢ . . . ?while ⦃´?inv ∧ ¬ (´M ≠ a)⦄
  proof
    let ?c = ´S :== ´S + b;; ´M :== ´M + 1
    have ⦃´?inv ∧ ´M ≠ a⦄ ⊆ ⦃´S + b = (´M + 1) ∗ b⦄
      by auto
    also have Γ⊢ . . . ?c ⦃´?inv⦄ by vcg
    finally show Γ⊢ ⦃´?inv ∧ ´M ≠ a⦄ ?c ⦃´?inv⦄ .
  qed
  also have ⦃´?inv ∧ ¬ (´M ≠ a)⦄ ⊆ ⦃´S = a ∗ b⦄ by auto
  finally show ?thesis by blast
qed
```

The subsequent version of the proof applies the *vcg* method to reduce the Hoare statement to a purely logical problem that can be solved fully automatically. Note that we have to specify the `WHILE` loop invariant in the original statement.

```
lemma (in state-space)
  shows Γ⊢ ⦃´M = 0 ∧ ´S = 0⦄
        WHILE ´M ≠ a
        INV ⦃´S = ´M ∗ b⦄
        DO ´S :== ´S + b;; ´M :== ´M + 1 OD
```

```
        ⦃´S = a ∗ b⦄
  apply vcg
  apply auto
  done
```

Here some examples of “breaking” out of a loop

```
lemma (in state-space)
  shows Γ⊢ ⦃´M = 0 ∧ ´S = 0⦄
        TRY
          WHILE True
          INV ⦃´S = ´M ∗ b⦄
          DO IF ´M = a THEN THROW ELSE ´S :== ´S + b;; ´M :== ´M +
1 FI OD
        CATCH
          SKIP
        END
        ⦃´S = a ∗ b⦄
apply vcg
apply auto
done

lemma (in state-space)
  shows Γ⊢ ⦃´M = 0 ∧ ´S = 0⦄
        TRY
          WHILE True
          INV ⦃´S = ´M ∗ b⦄
          DO IF ´M = a THEN ´Abr :== ''Break'';;THROW
             ELSE ´S :== ´S + b;; ´M :== ´M + 1
             FI
          OD
        CATCH
          IF ´Abr = ''Break'' THEN SKIP ELSE Throw FI
        END
        ⦃´S = a ∗ b⦄
apply vcg
apply auto
done
```

Some more syntactic sugar, the label statement ... · ... as shorthand for the *TRY−CATCH* above, and the *RAISE* for an state-update followed by a *THROW*.

```
lemma (in state-space)
  shows Γ⊢ ⦃´M = 0 ∧ ´S = 0⦄
        ⦃´Abr = ''Break''⦄· WHILE True INV ⦃´S = ´M ∗ b⦄
         DO IF ´M = a THEN RAISE ´Abr :== ''Break''
            ELSE ´S :== ´S + b;; ´M :== ´M + 1
            FI
         OD
        ⦃´S = a ∗ b⦄
```

apply *vcg*
apply *auto*
done

lemma (**in** *state-space*)
  shows $\Gamma\vdash$ $\{\!|\acute{}M = 0 \wedge \acute{}S = 0|\!\}$
        TRY
          WHILE True
          INV $\{\!|\acute{}S = \acute{}M * b|\!\}$
          DO IF ´M = a THEN RAISE ´Abr :== ''Break''
            ELSE ´S :== ´S + b;; ´M :== ´M + 1
            FI
          OD
        CATCH
          IF ´Abr = ''Break'' THEN SKIP ELSE Throw FI
        END
        $\{\!|\acute{}S = a * b|\!\}$
apply *vcg*
apply *auto*
done

lemma (**in** *state-space*)
  shows $\Gamma\vdash$ $\{\!|\acute{}M = 0 \wedge \acute{}S = 0|\!\}$
        $\{\!|\acute{}Abr = ''Break''|\!\}$ $\cdot$ *WHILE True*
        INV $\{\!|\acute{}S = \acute{}M * b|\!\}$
        DO IF ´M = a THEN RAISE ´Abr :== ''Break''
            ELSE ´S :== ´S + b;; ´M :== ´M + 1
            FI
        OD
        $\{\!|\acute{}S = a * b|\!\}$
apply *vcg*
apply *auto*
done

Blocks

lemma (**in** *state-space*)
  shows $\Gamma\vdash\{\!|\acute{}I = i|\!\}$ *LOC ´I;; ´I :== 2 COL* $\{\!|\acute{}I \leq i|\!\}$
  apply *vcg*
  by *simp*

18.4 Summing Natural Numbers

We verify an imperative program to sum natural numbers up to a given limit. First some functional definition for proper specification of the problem.

primrec
  sum :: (*nat* => *nat*) => *nat* => *nat*
where
  sum f 0 = *0*

| *sum f* (*Suc n*) = *f n* + *sum f n*

```
syntax
  -sum :: idt => nat => nat => nat
    (SUMM -<-. - [0, 0, 10] 10)
translations
  SUMM j<k. b == CONST sum (λj. b) k
```

The following proof is quite explicit in the individual steps taken, with the *vcg* method only applied locally to take care of assignment and sequential composition. Note that we express intermediate proof obligation in pure logic, without referring to the state space.

```
theorem (in state-space)
  shows Γ⊢ {|True|}
           ´S :== 0;; ´I :== 1;;
           WHILE ´I ≠ n
           DO
             ´S :== ´S + ´I;;
             ´I :== ´I + 1
           OD
           {|´S = (SUMM j<n. j)|}
  (is Γ⊢ - (-;; ?while) -)
proof −
  let ?sum = λk. SUMM j<k. j
  let ?inv = λs i. s = ?sum i

  have Γ⊢ {|True|} ´S :== 0;; ´I :== 1 {|?inv ´S ´I|}
  proof −
    have True ⟶ 0 = ?sum 1
      by simp
    also have Γ⊢ {|...|} ´S :== 0;; ´I :== 1 {|?inv ´S ´I|}
      by vcg
    finally show ?thesis .
  qed
  also have Γ⊢ {|?inv ´S ´I|} ?while {|?inv ´S ´I ∧ ¬ ´I ≠ n|}
  proof
    let ?body = ´S :== ´S + ´I;; ´I :== ´I + 1
    have ⋀s i. ?inv s i ∧ i ≠ n ⟶  ?inv (s + i) (i + 1)
      by simp
    also have Γ⊢ {|´S + ´I = ?sum (´I + 1)|} ?body {|?inv ´S ´I|}
      by vcg
    finally show Γ⊢ {|?inv ´S ´I ∧ ´I ≠ n|} ?body {|?inv ´S ´I|} .
  qed
  also have ⋀s i. s = ?sum i ∧ ¬ i ≠ n ⟶ s = ?sum n
    by simp
  finally show ?thesis .
qed
```

The next version uses the *vcg* method, while still explaining the resulting proof obligations in an abstract, structured manner.

theorem (**in** *state-space*)
  shows $\Gamma \vdash \{\!|\mathit{True}|\!\}$
    $\acute{}S :== 0;;\ \acute{}I :== 1;;$
    $\mathit{WHILE}\ \acute{}I \neq n$
    $\mathit{INV}\ \{\!|\acute{}S = (\mathit{SUMM}\ j{<}\acute{}I.\ j)|\!\}$
    DO
      $\acute{}S :== \acute{}S + \acute{}I;;$
      $\acute{}I :== \acute{}I + 1$
    OD
    $\{\!|\acute{}S = (\mathit{SUMM}\ j{<}n.\ j)|\!\}$
proof −
  let $?sum = \lambda k.\ \mathit{SUMM}\ j{<}k.\ j$
  let $?inv = \lambda s\ i.\ s = ?sum\ i$

  show *?thesis*
  proof *vcg*
    show *?inv 0 1* **by** *simp*
  next
    fix *i s* **assume** $?inv\ s\ i\ i \neq n$
    thus $?inv\ (s + i)\ (i + 1)$ **by** *simp*
  next
    fix *i s* **assume** x: $?inv\ s\ i\ \neg\ i \neq n$
    thus $s = ?sum\ n$ **by** *simp*
  qed
qed

Certainly, this proof may be done fully automatically as well, provided that the invariant is given beforehand.

theorem (**in** *state-space*)
  shows $\Gamma \vdash \{\!|\mathit{True}|\!\}$
    $\acute{}S :== 0;;\ \acute{}I :== 1;;$
    $\mathit{WHILE}\ \acute{}I \neq n$
    $\mathit{INV}\ \{\!|\acute{}S = (\mathit{SUMM}\ j{<}\acute{}I.\ j)|\!\}$
    DO
      $\acute{}S :== \acute{}S + \acute{}I;;$
      $\acute{}I :== \acute{}I + 1$
    OD
    $\{\!|\acute{}S = (\mathit{SUMM}\ j{<}n.\ j)|\!\}$
  apply *vcg*
  apply *auto*
  done

18.5 SWITCH

lemma (**in** *state-space*)
  shows $\Gamma \vdash \{\!|\acute{}N = 5|\!\}\ \mathit{SWITCH}\ \acute{}B$
      $\{\mathit{True}\} \Rightarrow \acute{}N :== 6$
    $|\ \{\mathit{False}\} \Rightarrow \acute{}N :== 7$

$\quad\quad END$
$\quad \{\!|\, ´N > 5 \,|\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *state-space*)
shows $\Gamma\vdash \{\!|\, ´N = 5 \,|\!\}$ *SWITCH ´N*
$\quad\quad \{v.\ v < 5\} \Rightarrow ´N :== 6$
$\quad\quad |\ \{v.\ v \geq 5\} \Rightarrow ´N :== 7$
$\quad\quad END$
$\quad \{\!|\, ´N > 5 \,|\!\}$
apply *vcg*
apply *simp*
done

18.6 (Mutually) Recursive Procedures

18.6.1 Factorial

We want to define a procedure for the factorial. We first define a HOL functions that calculates it to specify the procedure later on.

primrec *fac*:: $nat \Rightarrow nat$
where
$fac\ 0 = 1\ |$
$fac\ (Suc\ n) = (Suc\ n) * fac\ n$

lemma *fac-simp* [*simp*]: $0 < i \Longrightarrow fac\ i = i * fac\ (i - 1)$
by (*cases i*) *simp-all*

Now we define the procedure

procedures
Fac (*N*::*nat*|*R*::*nat*)
IF ´N = 0 THEN ´R :== 1
ELSE ´R :== CALL Fac(´N − 1);;
´R :== ´N ∗ ´R
FI

print-locale *Fac-impl*

To see how a call is syntactically translated you can switch off the printing translation via the configuration option *hoare-use-call-tr′*

context *Fac-impl*
begin

´R :== CALL Fac(´N) is internally:

declare [[*hoare-use-call-tr′* = *false*]] *call* ($\lambda s.\ s(\!|$*locals* := *update project-Nat-nat inject-Nat-nat N-'Fac-'* (*K-statefun* (*lookup project-Nat-nat N-'Fac-'* (*locals s*))) (*locals s*)$|\!)$) *Fac-'proc* ($\lambda s\ t.$ $s(\!|$*globals* := *globals t*$|\!)$) ($\lambda i\ t.$ *´R* :== *lookup project-Nat-nat R-'Fac-'* (*locals t*))

term *CALL Fac*(*´N*,*´R*)
declare [[*hoare-use-call-tr' = true*]]

Now let us prove that *Fac* meets its specification.

end

lemma (**in** *Fac-impl*) *Fac-spec'*:
  shows $\forall \sigma.\ \Gamma,\Theta\vdash\{\sigma\}$ *PROC Fac*(*´N*,*´R*) $\{\!|$*´R = fac* $^{\sigma}N|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *simp*
  done

Since the factorial was implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from general rule for mutually recursive procedures):

$[\![\forall Z.\ \Gamma,\Theta \cup (\bigcup_Z \{(P\ Z,\ p,\ Q\ Z,\ A\ Z)\})\vdash_{/F} (P\ Z)\ \mathit{the}\ (\Gamma\ p)\ (Q\ Z),(A\ Z);$
$p \in \mathit{dom}\ \Gamma]\!]$
$\Longrightarrow \forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ \mathit{Call}\ p\ (Q\ Z),(A\ Z)$

The verification condition generator will infer the specification out of the context when it encounters a recursive call of the factorial.

We can also step through verification condition generation. When the verification condition generator encounters a procedure call it tries to use the rule *ProcSpec*. To be successful there must be a specification of the procedure in the context.

lemma (**in** *Fac-impl*) *Fac-spec1*:
  shows $\forall \sigma.\ \Gamma,\Theta\vdash\{\sigma\}$ *´R* :== *PROC Fac*(*´N*) $\{\!|$*´R = fac* $^{\sigma}N|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *vcg-step*
  apply *simp*
  done

Here some Isar style version of the proof

lemma (**in** *Fac-impl*) *Fac-spec2*:

shows $\forall\sigma.\ \Gamma,\Theta\vdash\{\sigma\}$ *´R :== PROC Fac(´N)* $\{\!|$ *´R = fac* $^{\sigma}N|\!\}$
proof (*hoare-rule HoarePartial.ProcRec1*)
have *Fac-spec*: $\forall\sigma.\ \Gamma,(\Theta\cup(\bigcup\sigma.\ \{(\{\sigma\},\ Fac\text{-}'proc,\ \{\!|$ *´R = fac* $^{\sigma}N|\!\},\{\})\}))$
$\vdash\{\sigma\}$ *´R :== PROC Fac(´N)* $\{\!|$ *´R = fac* $^{\sigma}N|\!\}$
apply (*rule allI*)
apply (*rule hoarep.Asm*)
by *simp*
show $\forall\sigma.\ \Gamma,(\Theta\cup(\bigcup\sigma.\ \{(\{\sigma\},\ Fac\text{-}'proc,\ \{\!|$ *´R = fac* $^{\sigma}N|\!\},\{\})\}))$
$\vdash\{\sigma\}$ *IF ´N = 0 THEN ´R :== 1*
ELSE ´R :== CALL Fac(´N − 1);; ´R :== ´N ∗ ´R FI $\{\!|$ *´R = fac* $^{\sigma}N|\!\}$
apply *vcg*
apply *simp*
done
qed

To avoid retyping of potentially large pre and postconditions in the previous proof we can use the casual term abbreviations of the Isar language.

lemma (**in** *Fac-impl*) *Fac-spec3*:
shows $\forall\sigma.\ \Gamma,\Theta\vdash\{\sigma\}$ *´R :== PROC Fac(´N)* $\{\!|$ *´R = fac* $^{\sigma}N|\!\}$
(**is** $\forall\sigma.\ \Gamma,\Theta\vdash$(*?Pre σ*) *?Fac* (*?Post σ*))
proof (*hoare-rule HoarePartial.ProcRec1*)
have *Fac-spec*: $\forall\sigma.\ \Gamma,(\Theta\cup(\bigcup\sigma.\ \{($*?Pre σ*, *Fac-'proc*, *?Post σ*,$\{\})\}))$
$\vdash$(*?Pre σ*) *?Fac* (*?Post σ*)
apply (*rule allI*)
apply (*rule hoarep.Asm*)
by *simp*
show $\forall\sigma.\ \Gamma,(\Theta\cup(\bigcup\sigma.\ \{($*?Pre σ*, *Fac-'proc*, *?Post σ*,$\{\})\}))$
$\vdash$ (*?Pre σ*) *IF ´N = 0 THEN ´R :== 1*
ELSE ´R :== CALL Fac(´N − 1);; ´R :== ´N ∗ ´R FI (*?Post σ*)
apply *vcg*
apply *simp*
done
qed

The previous proof pattern has still some kind of inconvenience. The augmented context is always printed in the proof state. That can mess up the state, especially if we have large specifications. This may be annoying if we want to develop single step or structured proofs. In this case it can be a good idea to introduce a new variable for the augmented context.

lemma (**in** *Fac-impl*) *Fac-spec4*:
shows $\forall\sigma.\ \Gamma,\Theta\vdash\{\sigma\}$ *´R :== PROC Fac(´N)* $\{\!|$ *´R = fac* $^{\sigma}N|\!\}$
(**is** $\forall\sigma.\ \Gamma,\Theta\vdash$(*?Pre σ*) *?Fac* (*?Post σ*))
proof (*hoare-rule HoarePartial.ProcRec1*)
define Θ' **where** $\Theta' = \Theta\cup(\bigcup\sigma.\ \{($*?Pre σ*, *Fac-'proc*, *?Post σ*,$\{\})\})$
have *Fac-spec*: $\forall\sigma.\ \Gamma,\Theta'\vdash$(*?Pre σ*) *?Fac* (*?Post σ*)
by (*unfold Θ'-def*, *rule allI*, *rule hoarep.Asm*) *simp*

We have to name the fact *Fac-spec*, so that the vcg can use the specification for the recursive call, since it cannot infer it from the opaque Θ'.

show $\forall \sigma.\ \Gamma,\Theta' \vdash$ *(?Pre σ) IF ´N = 0 THEN ´R :== 1*
ELSE ´R :== CALL Fac(´N − 1);; ´R :== ´N ∗ ´R FI (?Post σ)
apply *vcg*
apply *simp*
done
qed

There are different rules available to prove procedure calls, depending on the kind of postcondition and whether or not the procedure is recursive or even mutually recursive. See for example *HoareTotal.ProcRec1*, *HoareTotal.ProcNoRec1*. They are all derived from the most general rule *HoareTotal.ProcRec*. All of them have some side-conditions concerning the parameter passing protocol and its relation to the pre and postcondition. They can be solved in a uniform fashion. Thats why we have created the method *hoare-rule*, which behaves like the method *rule* but automatically tries to solve the side-conditions.

18.6.2 Odd and Even

Odd and even are defined mutually recursive here. In the *procedures* command we conjoin both definitions with *and*.

procedures
odd(N::nat | A::nat) IF ´N=0 THEN ´A:==0
ELSE IF ´N=1 THEN CALL even (´N − 1,´A)
ELSE CALL odd (´N − 2,´A)
FI
FI

and
even(N::nat | A::nat) IF ´N=0 THEN ´A:==1
ELSE IF ´N=1 THEN CALL odd (´N − 1,´A)
ELSE CALL even (´N − 2,´A)
FI
FI
print-theorems
print-locale! *odd-even-clique*

To prove the procedure calls to *odd* respectively *even* correct we first derive a rule to justify that we can assume both specifications to verify the bodies. This rule can be derived from the general *HoareTotal.ProcRec* rule. An ML function will do this work:

ML ‹*ML-Thms.bind-thm (ProcRec2, Hoare.gen-proc-rec @{context} Hoare.Partial 2)*›

lemma (**in** *odd-even-clique*)
 shows *odd-spec*: $\forall\sigma.\ \Gamma\vdash\{\sigma\}$ $\acute{}A$:== *PROC odd*($\acute{}N$)
 $\{\!|(\exists\, b.\ {}^{\sigma}N = 2 * b + \acute{}A) \wedge \acute{}A < 2\ |\!\}$ (**is** *?P1*)
 and *even-spec*: $\forall\sigma.\ \Gamma\vdash\{\sigma\}$ $\acute{}A$:== *PROC even*($\acute{}N$)
 $\{\!|(\exists\, b.\ {}^{\sigma}N + 1 = 2 * b + \acute{}A) \wedge \acute{}A < 2\ |\!\}$ (**is** *?P2*)
proof −
 have *?P1* $\wedge$ *?P2*
 apply (*hoare-rule ProcRec2*)
 apply *vcg*
 apply *clarsimp*
 apply (*rule-tac x=b + 1* **in** *exI*)
 apply *arith*
 apply *vcg*
 apply *clarsimp*
 apply *arith*
 done
 thus *?P1 ?P2*
 by *iprover+*
qed

18.7 Expressions With Side Effects

lemma (**in** *state-space*) **shows** $\Gamma\vdash$ $\{\!|$*True*$|\!\}$
 $\acute{}N \gg n.\ \acute{}N$:== $\acute{}N + 1 \gg$
 $\acute{}M \gg m.\ \acute{}M$:== $\acute{}M + 1 \gg$
 $\acute{}R$:== $n + m$
 $\{\!|\acute{}R = \acute{}N + \acute{}M - 2|\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *Fac-impl*) **shows**
 $\Gamma\vdash$ $\{\!|$*True*$|\!\}$
 CALL Fac($\acute{}N$) $\gg n.$ *CALL Fac*($\acute{}N$) $\gg m.$
 $\acute{}R$:== $n + m$
 $\{\!|\acute{}R =$ *fac* $\acute{}N$ + *fac* $\acute{}N|\!\}$
proof −
 note *Fac-spec* = *Fac-spec4*
 show *?thesis*
 apply *vcg*
 done
qed

lemma (**in** *Fac-impl*) **shows**
 $\Gamma\vdash$ $\{\!|$*True*$|\!\}$
 CALL Fac($\acute{}N$) $\gg n.$ *CALL Fac*(n) $\gg m.$

´*R* :== *m*
⦃´*R* = *fac* (*fac* ´*N*)⦄
proof −
note *Fac-spec* = *Fac-spec4*
show *?thesis*
apply *vcg*
done
qed

18.8 Global Variables and Heap

Now we will define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

hoarestate *globals-list* =
next :: *ref* ⇒ *ref*
cont :: *ref* ⇒ *nat*

Updates to global components inside a procedure will always be propagated to the caller. This is implicitly done by the parameter passing syntax translations. The record containing the global variables must begin with the prefix "globals".

We will first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter, and returns the result right into the first parameter.

procedures (**imports** *globals-list*)
append(*p*::*ref*,*q*::*ref*|*p*::*ref*)
IF ´*p*=*Null THEN* ´*p* :== ´*q ELSE* ´*p* →´*next*:== *CALL append*(´*p*→´*next*,´*q*) *FI*

declare [[*hoare-use-call-tr′* = *false*]]
context *append-impl*
begin
term *CALL append*(´*p*,´*q*,´*p*→´*next*)
end
declare [[*hoare-use-call-tr′* = *true*]]

Below we give two specifications this time.. The first one captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the second one is a pure frame condition. The list in the modifies clause has to list all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts of the lists will not be changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state t here.

The functional specification now introduces two logical variables besides the state space variable σ, namely *Ps* and *Qs*. They are universally quantified and range over both the pre and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax $\{\!|\sigma.\ \ldots|\!\}$ is a shorthand to fix the current state: $\{s.\ \sigma = s\ \ldots\}$.

lemma (**in** *append-impl*) *append-spec*:
  shows $\forall\sigma\ Ps\ Qs.\ \Gamma\vdash$
  $\{\!|\sigma.\ List\ \acute{}p\ \acute{}next\ Ps \wedge\ List\ \acute{}q\ \acute{}next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}|\!\}$
  $\acute{}p :== PROC\ append(\acute{}p,\acute{}q)$
  $\{\!|List\ \acute{}p\ \acute{}next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow \acute{}next\ x = {}^{\sigma}next\ x)|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *fastforce*
  done

The modifies clause is equal to a proper record update specification of the following form.

lemma (**in** *append-impl*) **shows** $\{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ Z\ in\ [next]\}$
  $=$
  $\{t.\ \exists next.\ globals\ t{=}update\ id\ id\ next\text{-}'\ (K\text{-}statefun\ next)\ (globals\ Z)\}$
  apply (*unfold mex-def meq-def*)
  apply *simp*
  done

If the verification condition generator works on a procedure call it checks whether it can find a modifies clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoareTotal.ProcSpec* is applied. Simplification of the procedure call means, that the "copy back" of the global components is simplified. Only those components that occur in the modifies clause will actually be copied back. This simplification is justified by the rule *HoareTotal.ProcModifyReturn*. So after this simplification all global components that do not appear in the modifies clause will be treated as local variables.

You can study the effect of the modifies clause on the following two examples, where we want to prove that (@) does not change the *cont* part of the heap.

lemma (**in** *append-impl*)
  shows $\Gamma\vdash \{\!|\acute{}p{=}Null \wedge \acute{}cont{=}c|\!\}\ \acute{}p :== CALL\ append(\acute{}p,Null)\ \{\!|\acute{}cont{=}c|\!\}$
  apply *vcg*
  oops

To prove the frame condition, we have to tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies* It will also try to solve the verification conditions automatically.

lemma (**in** *append-impl*) *append-modifies*:
shows
$\forall\sigma.\ \Gamma\vdash\ \{\sigma\}$ *´p :== PROC append(´p,´q)*$\{t.$ *t may-only-modify-globals* σ *in* [*next*]$\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma (**in** *append-impl*)
shows $\Gamma\vdash$ {|*´p=Null* $\wedge$ *´cont=c*|} *´p→´next :== CALL append(´p,Null)* {|*´cont=c*|}
apply *vcg*
apply *simp*
done

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To verify the body of (@) we do not need the modifies clause, since the specification does not talk about *cont* at all, and we don't access *cont* inside the body. This may be different for more complex procedures.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but me may need the modifies clause to prove the functional specifications.

18.8.1 Insertion Sort

primrec *sorted*:: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list\ \Rightarrow bool$
where
sorted le [] = *True* |
sorted le (*x#xs*) = (($\forall$ *y*$\in$*set xs. le x y*) $\wedge$ *sorted le xs*)

procedures (**imports** *globals-list*)
insert(*r*::*ref*,*p*::*ref* | *p*::*ref*)
IF ´r=Null THEN SKIP
ELSE IF ´p=Null THEN ´p :== ´r;; ´p→´next :== Null
ELSE IF ´r→´cont ≤ ´p→´cont
THEN ´r→´next :== ´p;; ´p:==´r
ELSE ´p→´next :== CALL insert(´r,´p→´next)
FI

FI
FI

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state, even in the conclusion of the implication. The reason is, that we have separated out, that *cont* is not modified by the procedure, to the modifies clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitely state an equality in the postcondition. The reason is simple. If the postcondition would talk about *´cont* instead of $^{\sigma}$*cont*, we will get a new instance of *cont* during verification and the postcondition would only state something about this new instance. But as the verification condition generator will use the modifies clause the caller of *insert* instead will still have the old *cont* after the call. Thats the sense of the modifies clause. So the caller and the specification will simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

lemma (**in** *insert-impl*) *insert-modifies*:
$\forall \sigma.\ \Gamma\vdash \{\sigma\}\ {´p} :== \mathit{PROC\ insert}({´r},{´p})\{t.\ t\ \mathit{may\text{-}only\text{-}modify\text{-}globals}\ \sigma\ \mathit{in}\ [\mathit{next}]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

lemma (**in** *insert-impl*) *insert-spec*:
$\forall \sigma\ Ps\ .\ \Gamma\vdash \{\!|\sigma.\ \mathit{List}\ {´p}\ {´next}\ Ps \wedge \mathit{sorted}\ (\leq)\ (\mathit{map}\ {´cont}\ Ps) \wedge$
$\quad {´r} \neq \mathit{Null} \wedge {´r} \notin \mathit{set}\ Ps|\!\}$
$\quad {´p} :== \mathit{PROC\ insert}({´r},{´p})$
$\{\!|\exists\ Qs.\ \mathit{List}\ {´p}\ {´next}\ Qs \wedge \mathit{sorted}\ (\leq)\ (\mathit{map}\ {}^{\sigma}\mathit{cont}\ \ Qs) \wedge$
$\quad \mathit{set}\ Qs = \mathit{insert}\ {}^{\sigma}r\ (\mathit{set}\ Ps) \wedge$
$\quad (\forall x.\ x \notin \mathit{set}\ Qs \longrightarrow {´next}\ x = {}^{\sigma}\mathit{next}\ x)|\!\}$

apply (*hoare-rule HoarePartial.ProcRec1*)
apply *vcg*
apply (*intro conjI impI*)
apply *fastforce*
apply *fastforce*
apply *fastforce*
apply (*clarsimp*)
apply *force*
done

procedures (**imports** *globals-list*)
insertSort(*p*::*ref* | *p*::*ref*)
where *r*::*ref* *q*::*ref*
in
´r:==*Null*;;
WHILE (*´p* ≠ *Null*) *DO*

```
      ´q :== ´p;;
      ´p :== ´p→´next;;
      ´r :== CALL insert(´q,´r)
    OD;;
    ´p:==´r
```

print-locale *insertSort-impl*

lemma (**in** *insertSort-impl*) *insertSort-modifies*:
  shows
   $\forall\sigma.\ \Gamma\vdash \{\sigma\}$ *´p :== PROC insertSort(´p)*
     $\{t.\ t$ *may-only-modify-globals* σ *in* $[next]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

Insertion sort is not implemented recursively here but with a while loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. Therefore we will annotate the body during the proof with the rule *HoareTotal.annotateI*.

```
lemma (in insertSort-impl) insertSort-body-spec:
  shows ∀σ Ps. Γ,Θ⊢ ⦃σ. List ´p ´next Ps ⦄
            ´p :== PROC insertSort(´p)
        ⦃∃Qs. List ´p ´next Qs ∧ sorted (≤) (map σcont Qs) ∧
         set Qs = set Ps⦄
  apply (hoare-rule HoarePartial.ProcRec1)
  apply (hoare-rule anno=
        ´r :== Null;;
        WHILE ´p ≠ Null
        INV ⦃∃Qs Rs. List ´p ´next Qs ∧ List ´r ´next Rs ∧
                set Qs ∩ set Rs = {} ∧
                sorted (≤) (map ´cont Rs) ∧ set Qs ∪ set Rs = set Ps ∧
                ´cont = σcont ⦄
         DO ´q :== ´p;; ´p :== ´p→´next;; ´r :== CALL insert(´q,´r) OD;;
         ´p :== ´r in HoarePartial.annotateI)
  apply vcg
  apply  fastforce
  prefer 2
  apply  fastforce
  apply (clarsimp)
  apply (rule-tac x=ps in exI)
  apply (intro conjI)
  apply    (rule heap-eq-ListI1)
  apply     assumption
  apply    clarsimp
  apply    (subgoal-tac x≠p ∧ x ∉ set Rs)
  apply     auto
```

done

18.8.2 Memory Allocation and Deallocation

The basic idea of memory management is to keep a list of allocated references in the state space. Allocation of a new reference adds a new reference to the list deallocation removes a reference. Moreover we keep a counter "free" for the free memory.

hoarestate *globals-list-alloc* =
  alloc::*ref list*
  free::*nat*
  next::*ref* ⇒ *ref*
  cont::*ref* ⇒ *nat*
hoarestate *locals-list-alloc* =
  i::*nat*
  first::*ref*
  p::*ref*
  q::*ref*
  r::*ref*
  root::*ref*
  tmp::*ref*
locale *list-alloc* = *globals-list-alloc* + *locals-list-alloc*

definition *sz* = (*2*::*nat*)

lemma (**in** *list-alloc*)
 shows
  Γ,Θ⊢ {|´*i* = *0* ∧ ´*first* = *Null* ∧ *n*∗*sz* ≤ ´*free*|}
   WHILE ´*i* < *n*
   INV {|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = ´*i* ∧ ´*i* ≤ *n* ∧
     set Ps ⊆ *set* ´*alloc* ∧ (*n* − ´*i*)∗*sz* ≤ ´*free*|}
   DO
    ´*p* :== *NEW sz* [´*cont*:==*0*,´*next*:== *Null*];;
    ´*p*→´*next* :== ´*first*;;
    ´*first* :== ´*p*;;
    ´*i* :== ´*i*+ *1*
   OD
   {|∃ *Ps*. *List* ´*first* ´*next* *Ps* ∧ *length Ps* = *n* ∧ *set Ps* ⊆ *set* ´*alloc*|}
apply (*vcg*)
apply *simp*
apply *clarsimp*
apply (*rule conjI*)
apply *clarsimp*
apply (*rule-tac x*=*new* (*set alloc*)#*Ps* **in** *exI*)
apply *clarsimp*
apply (*rule conjI*)
apply *fastforce*
apply (*simp add*: *sz-def*)

apply (*simp add*: *sz-def*)
apply *fastforce*
done

lemma (**in** *list-alloc*)
shows
$\Gamma\vdash$ {|´*i* = *0* ∧ ´*first* = *Null* ∧ *n*∗*sz* ≤ ´*free*|}
WHILE ´*i* < *n*
INV {|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = ´*i* ∧ ´*i* ≤ *n* ∧
set Ps ⊆ *set* ´*alloc* ∧ (*n* − ´*i*)∗*sz* ≤ ´*free*|}
DO
´*p* :== *NNEW sz* [´*cont*:==*0*,´*next*:== *Null*];;
´*p*→´*next* :== ´*first*;;
´*first* :== ´*p*;;
´*i* :== ´*i*+ *1*
OD
{|∃ *Ps*. *List* ´*first* ´*next Ps* ∧ *length Ps* = *n* ∧ *set Ps* ⊆ *set* ´*alloc*|}

apply (*vcg*)
apply *simp*
apply *clarsimp*
apply (*rule conjI*)
apply *clarsimp*
apply (*rule-tac x*=*new* (*set alloc*)#*Ps* **in** *exI*)
apply *clarsimp*
apply (*rule conjI*)
apply *fastforce*
apply (*simp add*: *sz-def*)
apply (*simp add*: *sz-def*)
apply *fastforce*
done

18.9 Fault Avoiding Semantics

If we want to ensure that no runtime errors occur we can insert guards into the code. We will not be able to prove any nontrivial Hoare triple about code with guards, if we cannot show that the guards will never fail. A trivial Hoare triple is one with an empty precondtion.

lemma (**in** *list-alloc*) Γ,Θ⊢ {|*True*|} {|´*p*≠*Null*|}⟼ ´*p*→´*next* :== ´*p* {|*True*|}
apply *vcg*
oops

lemma (**in** *list-alloc*) Γ,Θ⊢ {} {|´*p*≠*Null*|}⟼ ´*p*→´*next* :== ´*p* {|*True*|}
apply *vcg*
done

Let us consider this small program that reverts a list. At first without guards.

lemma (**in** *list-alloc*)
shows
$\Gamma,\Theta\vdash$ $\{\!|$*List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* = {} $\wedge$
set Ps $\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
WHILE ´p $\neq$ *Null*
INV $\{\!|\exists$ *ps qs. List ´p ´next ps* $\wedge$ *List ´q ´next qs* $\wedge$ *set ps* $\cap$ *set qs* = {} $\wedge$
rev ps @ qs = rev Ps @ Qs $\wedge$
set ps $\subseteq$ *set ´alloc* $\wedge$ *set qs* $\subseteq$ *set ´alloc*$|\!\}$
DO ´r :== ´p;;
´p :== ´p→ ´next;;
´r→´next :== ´q;;
´q :== ´r OD
$\{\!|$*List ´q ´next (rev Ps @ Qs)* $\wedge$ *set Ps*$\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
apply (*vcg*)
apply *fastforce+*
done

If we want to ensure that we do not dereference *Null* or access unallocated memory, we have to add some guards.

lemma (**in** *list-alloc*)
shows
$\Gamma,\Theta\vdash$ $\{\!|$*List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* = {} $\wedge$
set Ps $\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
WHILE ´p $\neq$ *Null*
INV $\{\!|\exists$ *ps qs. List ´p ´next ps* $\wedge$ *List ´q ´next qs* $\wedge$ *set ps* $\cap$ *set qs* = {} $\wedge$
rev ps @ qs = rev Ps @ Qs $\wedge$
set ps $\subseteq$ *set ´alloc* $\wedge$ *set qs* $\subseteq$ *set ´alloc*$|\!\}$
DO ´r :== ´p;;
$\{\!|$*´p*$\neq$*Null* $\wedge$ *´p*$\in$*set ´alloc*$|\!\}\longmapsto$ *´p :== ´p→ ´next;;*
$\{\!|$*´r*$\neq$*Null* $\wedge$ *´r*$\in$*set ´alloc*$|\!\}\longmapsto$ *´r→´next :== ´q;;*
´q :== ´r OD
$\{\!|$*List ´q ´next (rev Ps @ Qs)* $\wedge$ *set Ps* $\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
apply (*vcg*)
apply *fastforce+*
done

We can also just prove that no faults will occur, by giving the trivial postcondition.

lemma (**in** *list-alloc*) *rev-noFault*:
shows
$\Gamma,\Theta\vdash$ $\{\!|$*List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* = {} $\wedge$
set Ps $\subseteq$ *set ´alloc* $\wedge$ *set Qs* $\subseteq$ *set ´alloc*$|\!\}$
WHILE ´p $\neq$ *Null*
INV $\{\!|\exists$ *ps qs. List ´p ´next ps* $\wedge$ *List ´q ´next qs* $\wedge$ *set ps* $\cap$ *set qs* = {} $\wedge$
rev ps @ qs = rev Ps @ Qs $\wedge$
set ps $\subseteq$ *set ´alloc* $\wedge$ *set qs* $\subseteq$ *set ´alloc*$|\!\}$
DO ´r :== ´p;;
$\{\!|$*´p*$\neq$*Null* $\wedge$ *´p*$\in$*set ´alloc*$|\!\}\longmapsto$ *´p :== ´p→ ´next;;*
$\{\!|$*´r*$\neq$*Null* $\wedge$ *´r*$\in$*set ´alloc*$|\!\}\longmapsto$ *´r→´next :== ´q;;*

$\acute{}q$:== $\acute{}r$ *OD*
UNIV,*UNIV*
apply (*vcg*)
apply *fastforce+*
done

lemma (**in** *list-alloc*) *rev-moduloGuards*:

shows
$\Gamma,\Theta\vdash_{/\{True\}}$ $\{\!|$*List* $\acute{}p$ $\acute{}next$ *Ps* $\wedge$ *List* $\acute{}q$ $\acute{}next$ *Qs* $\wedge$ *set Ps* $\cap$ *set Qs* $= \{\} \wedge$
set Ps $\subseteq$ *set* $\acute{}alloc$ $\wedge$ *set Qs* $\subseteq$ *set* $\acute{}alloc|\!\}$
WHILE $\acute{}p \neq$ *Null*
INV $\{\!|\exists$ *ps qs*. *List* $\acute{}p$ $\acute{}next$ *ps* $\wedge$ *List* $\acute{}q$ $\acute{}next$ *qs* $\wedge$ *set ps* $\cap$ *set qs* $= \{\} \wedge$
rev ps @ *qs* = *rev Ps* @ *Qs* $\wedge$
set ps $\subseteq$ *set* $\acute{}alloc$ $\wedge$ *set qs* $\subseteq$ *set* $\acute{}alloc|\!\}$
DO $\acute{}r$:== $\acute{}p$;;
$\{\!|\acute{}p\neq Null \wedge \acute{}p\in set\ \acute{}alloc|\!\}\surd \longmapsto \acute{}p :== \acute{}p\rightarrow \acute{}next$;;
$\{\!|\acute{}r\neq Null \wedge \acute{}r\in set\ \acute{}alloc|\!\}\surd \longmapsto \acute{}r\rightarrow\acute{}next :== \acute{}q$;;
$\acute{}q$:== $\acute{}r$ *OD*
$\{\!|$*List* $\acute{}q$ $\acute{}next$ (*rev Ps* @ *Qs*) $\wedge$ *set Ps* $\subseteq$ *set* $\acute{}alloc$ $\wedge$ *set Qs* $\subseteq$ *set* $\acute{}alloc|\!\}$
apply *vcg*
apply *fastforce+*
done

lemma *CombineStrip'*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ *P c' Q*,*A*
assumes *deriv-strip*: $\Gamma,\Theta\vdash_{/\{\}}$ *P c'' UNIV*,*UNIV*
assumes *c''*: *c''*= *mark-guards False* (*strip-guards* ($-F$) *c'*)
assumes *c*: *c* = *mark-guards False c'*
shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q*,*A*
proof $-$
from *deriv-strip* [*simplified c''*]
have $\Gamma,\Theta\vdash$ *P* (*strip-guards* ($-$ *F*) *c'*) *UNIV*,*UNIV*
by (*rule HoarePartialProps.MarkGuardsD*)
with *deriv*
have $\Gamma,\Theta\vdash$ *P c' Q*,*A*
by (*rule HoarePartialProps.CombineStrip*)
hence $\Gamma,\Theta\vdash$ *P mark-guards False c' Q*,*A*
by (*rule HoarePartialProps.MarkGuardsI*)
thus *?thesis*
by (*simp add*: *c*)
qed

We can then combine the prove that no fault will occur with the functional prove of the programm without guards to get the full proove by the rule $[\![?\Gamma,?\Theta\vdash_{/?F}$ *?P ?c ?Q*,*?A*; $?\Gamma,?\Theta\vdash$ *?P strip-guards* ($-$ *?F*) *?c*

$UNIV, UNIV\rrbracket \Longrightarrow ?\Gamma, ?\Theta\vdash\ ?P\ ?c\ ?Q, ?A$

lemma (**in** *list-alloc*)
 shows
 $\Gamma,\Theta\vdash \{\!|List\ \acute{}p\ \acute{}next\ Ps \wedge List\ \acute{}q\ \acute{}next\ Qs \wedge set\ Ps \cap set\ Qs = \{\} \wedge$
 $set\ Ps \subseteq set\ \acute{}alloc \wedge set\ Qs \subseteq set\ \acute{}alloc|\!\}$
 $WHILE\ \acute{}p \neq Null$
 $INV\ \{\!|\exists ps\ qs.\ List\ \acute{}p\ \acute{}next\ ps \wedge List\ \acute{}q\ \acute{}next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$
 $rev\ ps\ @\ qs = rev\ Ps\ @\ Qs \wedge$
 $set\ ps \subseteq set\ \acute{}alloc \wedge set\ qs \subseteq set\ \acute{}alloc|\!\}$
 $DO\ \acute{}r :== \acute{}p;;$
 $\{\!|\acute{}p{\neq}Null \wedge \acute{}p{\in}set\ \acute{}alloc|\!\}{\longmapsto}\ \acute{}p :== \acute{}p{\rightarrow}\ \acute{}next;;$
 $\{\!|\acute{}r{\neq}Null \wedge \acute{}r{\in}set\ \acute{}alloc|\!\}{\longmapsto}\ \acute{}r{\rightarrow}\acute{}next :== \acute{}q;;$
 $\acute{}q :== \acute{}r\ OD$
 $\{\!|List\ \acute{}q\ \acute{}next\ (rev\ Ps\ @\ Qs) \wedge set\ Ps \subseteq set\ \acute{}alloc \wedge set\ Qs \subseteq set\ \acute{}alloc|\!\}$

apply (*rule CombineStrip′* [*OF rev-moduloGuards rev-noFault*])
apply *simp*
apply *simp*
done

In the previous example the effort to split up the prove did not really pay off. But when we think of programs with a lot of guards and complicated specifications it may be better to first focus on a prove without the messy guards. Maybe it is possible to automate the no fault proofs so that it suffices to focus on the stripped program.

context *list-alloc*
begin

The purpose of guards is to watch for faults that can occur during evaluation of expressions. In the example before we watched for null pointer dereferencing or memory faults. We can also look for array index bounds or division by zero. As the condition of a while loop is evaluated in each iteration we cannot just add a guard before the while loop. Instead we need a special guard for the condition. Example: $WHILE\ (False,\ \{\!|\acute{}p \neq Null|\!\}){\longmapsto}\ \acute{}p{\rightarrow}\acute{}next \neq Null\ DO\ SKIP\ OD$

end

18.10 Cicular Lists

definition
 $distPath :: ref \Rightarrow (ref \Rightarrow ref) \Rightarrow ref \Rightarrow ref\ list \Rightarrow bool$ **where**
 $distPath\ x\ next\ y\ as = (Path\ x\ next\ y\ as\ \wedge\ distinct\ as)$

lemma *neq-dP*: $\llbracket p \neq q;\ Path\ p\ h\ q\ Ps;\ distinct\ Ps\rrbracket \Longrightarrow$
 $\exists\, Qs.\ p{\neq}Null \wedge Ps = p\#Qs \wedge p \notin set\ Qs$
by (*cases Ps, auto*)

lemma (**in** *list-alloc*) *circular-list-rev-I*:
$\Gamma,\Theta\vdash$ $\{\!|$ *´root* $=$ *r* $\wedge$ *distPath ´root ´next ´root* (*r*#*Ps*)$|\!\}$
´p :== *´root*;; *´q* :== *´root*$\rightarrow$*´next*;;
WHILE ´q $\neq$ *´root*
INV $\{\!|$$\exists$ *ps qs*. *distPath ´p ´next ´root ps* $\wedge$ *distPath ´q ´next ´root qs* $\wedge$
´root $=$ *r* $\wedge$ *r*$\neq$*Null* $\wedge$ *r* $\notin$ *set Ps* $\wedge$ *set ps* $\cap$ *set qs* $=$ {} $\wedge$
Ps $=$ (*rev ps*) @ *qs* $|\!\}$
DO ´tmp :== *´q*;; *´q* :== *´q*$\rightarrow$*´next*;; *´tmp*$\rightarrow$*´next* :== *´p*;; *´p*:==*´tmp OD*;;
´root$\rightarrow$*´next* :== *´p*
$\{\!|$*´root* $=$ *r* $\wedge$ *distPath ´root ´next ´root* (*r*#*rev Ps*)$|\!\}$
apply (*simp only*:*distPath-def*)
apply *vcg*
apply (*rule-tac x*=[] **in** *exI*)
apply *fastforce*
apply *clarsimp*
apply (*drule* (*2*) *neq-dP*)
apply (*rule-tac x*=*q* # *ps* **in** *exI*)
apply *clarsimp*
apply *fastforce*
done

lemma *path-is-list*:$\bigwedge$*a next b*. $[\![$*Path b next a Ps* ; *a* $\notin$ *set Ps*; *a*$\neq$*Null*$]\!]$
$\Longrightarrow$ *List b* (*next*(*a* := *Null*)) (*Ps* @ [*a*])
apply (*induct Ps*)
apply (*auto simp add*:*fun-upd-apply*)
done

The simple algorithm for acyclic list reversal, with modified annotations, works for cyclic lists as well.:

lemma (**in** *list-alloc*) *circular-list-rev-II*:
$\Gamma,\Theta\vdash$
$\{\!|$*´p* $=$ *r* $\wedge$ *distPath ´p ´next ´p* (*r*#*Ps*)$|\!\}$
´q:==*Null*;;
WHILE ´p $\neq$ *Null*
INV
$\{\!|$ ((*´q* $=$ *Null*) $\longrightarrow$ ($\exists$*ps*. *distPath ´p ´next r ps* $\wedge$ *ps* $=$ *r*#*Ps*)) $\wedge$
((*´q* $\neq$ *Null*) $\longrightarrow$ ($\exists$*ps qs*. *distPath ´q ´next r qs* $\wedge$ *List ´p ´next ps* $\wedge$
set ps $\cap$ *set qs* $=$ {} $\wedge$ *rev qs* @ *ps* $=$ *Ps*@[*r*])) $\wedge$
$\neg$ (*´p* $=$ *Null* $\wedge$ *´q* $=$ *Null* $\wedge$ *r* $=$ *Null*)
$|\!\}$
DO
´tmp :== *´p*;; *´p* :== *´p*$\rightarrow$*´next*;; *´tmp*$\rightarrow$*´next* :== *´q*;; *´q*:==*´tmp*
OD
$\{\!|$*´q* $=$ *r* $\wedge$ *distPath ´q ´next ´q* (*r* # *rev Ps*)$|\!\}$

apply (*simp only*:*distPath-def*)

apply *vcg*
apply *clarsimp*
apply *clarsimp*
apply (*case-tac* ($q = Null$))
apply (*fastforce intro*: *path-is-list*)
apply *clarify*
apply (*rule-tac x=psa* **in** *exI*)
apply (*rule-tac x= p # qs* **in** *exI*)
apply *force*
apply *fastforce*
done

Although the above algorithm is more succinct, its invariant looks more involved. The reason for the case distinction on q is due to the fact that during execution, the pointer variables can point to either cyclic or acyclic structures.

When working on lists, its sometimes better to remove *fun-upd-apply* from the simpset, and instead include *fun-upd-same* and *fun-upd-other* to the simpset

lemma (**in** *state-space*) $\Gamma\vdash \{\sigma\}$
$\quad$ $\acute{I}$:== $\acute{M}$;;
$\quad$ *ANNO* τ. $\{\!|\tau.\ \acute{I} = {}^{\sigma}M|\!\}$
$\quad\quad$ $\acute{M}$:== $\acute{N}$;; $\acute{N}$:== $\acute{I}$
$\quad\quad$ $\{\!|\acute{M} = {}^{\tau}N \wedge \acute{N} = {}^{\tau}I|\!\}$
$\quad$ $\{\!|\acute{M} = {}^{\sigma}N \wedge \acute{N} = {}^{\sigma}M|\!\}$
apply *vcg*
apply *auto*
done

context *state-space*
begin
term *ANNO* (τ,m,k). ($\{\!|\tau.\ \acute{M} = m|\!\}$) $\acute{M}$:== $\acute{N}$;; $\acute{N}$:== $\acute{I}$ $\{\!|\acute{M} = {}^{\tau}N$ & $\acute{N} = {}^{\tau}I|\!\}$,$\{\}$
end

lemma (**in** *state-space*) $\Gamma\vdash$ ($\{\sigma\} \cap \{\!|\acute{M} = 0 \wedge \acute{S} = 0|\!\}$)
$\quad$ (*ANNO* τ. ($\{\tau\} \cap \{\!|\acute{A}={}^{\sigma}A \wedge \acute{I}={}^{\sigma}I \wedge \acute{M}=0 \wedge \acute{S}=0|\!\}$)
$\quad$ *WHILE* $\acute{M} \neq \acute{A}$
$\quad$ *INV* $\{\!|\acute{S} = \acute{M} * \acute{I} \wedge \acute{A}={}^{\tau}A \wedge \acute{I}={}^{\tau}I|\!\}$
$\quad$ *DO* $\acute{S}$:== $\acute{S} + \acute{I}$;; $\acute{M}$:== $\acute{M} + 1$ *OD*
$\quad$ $\{\!|\acute{S} = {}^{\tau}A * {}^{\tau}I|\!\}$)
$\quad$ $\{\!|\acute{S} = {}^{\sigma}A * {}^{\sigma}I|\!\}$
apply *vcg-step*
apply *vcg-step*
apply *simp*
apply *vcg-step*
apply *vcg-step*
apply *simp*

apply *vcg*
apply *simp*
apply *simp*
apply *vcg-step*
apply *auto*
done

Just some test on marked, guards

lemma (**in** *state-space*) $\Gamma \vdash \{\!| True |\!\}$ *WHILE* $\{\!| P\ \acute{}N\ |\!\}\surd, \{\!| Q\ \acute{}M |\!\}\#, \{\!| R\ \acute{}N |\!\} \longmapsto\ \acute{}N < \acute{}M$
INV $\{\!| \acute{}N < 2 |\!\}$ *DO*
$\acute{}N :== \acute{}M$
OD
$\{\!| hard |\!\}$
apply *vcg*
oops

lemma (**in** *state-space*) $\Gamma \vdash_{/\{True\}} \{\!| True |\!\}$ *WHILE* $\{\!| P\ \acute{}N\ |\!\}\surd, \{\!| Q\ \acute{}M |\!\}\#, \{\!| R\ \acute{}N |\!\} \longmapsto\ \acute{}N < \acute{}M$
INV $\{\!| \acute{}N < 2 |\!\}$ *DO*
$\acute{}N :== \acute{}M$
OD
$\{\!| hard |\!\}$
apply *vcg*
oops

end

19 Examples for Total Correctness

theory *VcgExTotal* **imports** ../*HeapList* ../*Vcg* **begin**

record *'g vars* = *'g state* +
A-' :: *nat*
I-' :: *nat*
M-' :: *nat*
N-' :: *nat*
R-' :: *nat*
S-' :: *nat*
Abr-':: *string*

lemma $\Gamma \vdash_t \{\!| \acute{}M = 0 \wedge \acute{}S = 0 |\!\}$
WHILE $\acute{}M \neq a$
INV $\{\!| \acute{}S = \acute{}M * b \wedge \acute{}M \leq a |\!\}$
VAR MEASURE $a - \acute{}M$
DO $\acute{}S :== \acute{}S + b;;\ \acute{}M :== \acute{}M + 1$ *OD*
$\{\!| \acute{}S = a * b |\!\}$

apply *vcg*
apply (*auto*)
done

lemma $\Gamma\vdash_t \{\!|\acute{}I \leq 3|\!\}$
$\quad$ *WHILE* $\acute{}I < 10$ *INV* $\{\!|\acute{}I \leq 10|\!\}$ *VAR MEASURE* $10 - \acute{}I$
$\quad$ *DO*
$\quad\quad$ $\acute{}I :== \acute{}I + 1$
$\quad$ *OD*
$\{\!|\acute{}I = 10|\!\}$
apply *vcg*
apply *auto*
done

Total correctness of a nested loop. In the inner loop we have to express that the loop variable of the outer loop is not changed. We use *FIX* to introduce a new logical variable

lemma $\Gamma\vdash_t \{\!|\acute{}M=0 \wedge \acute{}N=0|\!\}$
$\quad$ *WHILE* $(\acute{}M < i)$
$\quad$ *INV* $\{\!|\acute{}M \leq i \wedge (\acute{}M \neq 0 \longrightarrow \acute{}N = j) \wedge \acute{}N \leq j|\!\}$
$\quad$ *VAR MEASURE* $(i - \acute{}M)$
$\quad$ *DO*
$\quad\quad$ $\acute{}N :== 0;;$
$\quad\quad$ *WHILE* $(\acute{}N < j)$
$\quad\quad$ *FIX* $m.$
$\quad\quad$ *INV* $\{\!|\acute{}M=m \wedge \acute{}N \leq j|\!\}$
$\quad\quad$ *VAR MEASURE* $(j - \acute{}N)$
$\quad\quad$ *DO*
$\quad\quad\quad$ $\acute{}N :== \acute{}N + 1$
$\quad\quad$ *OD*;;
$\quad$ $\acute{}M :== \acute{}M + 1$
$\quad$ *OD*
$\quad$ $\{\!|\acute{}M=i \wedge (\acute{}M\neq 0 \longrightarrow \acute{}N=j)|\!\}$
apply *vcg*
apply *simp-all*
apply *arith*
done

primrec *fac*:: $nat \Rightarrow nat$
where
$fac\ 0 = 1\ |$
$fac\ (Suc\ n) = (Suc\ n) * fac\ n$

lemma *fac-simp* [*simp*]: $0 < i \Longrightarrow fac\ i = i * fac\ (i - 1)$
by (*cases i*) *simp-all*

procedures
$\quad$ $Fac\ (N\ |\ R) =$ *IF* $\acute{}N = 0$ *THEN* $\acute{}R :== 1$
$\quad\quad\quad\quad\quad\quad$ *ELSE CALL* $Fac(\acute{}N - 1, \acute{}R);;$

´R :== ´N ∗ ´R
FI

lemma (**in** *Fac-impl*) *Fac-spec*:
shows ∀ *n*. Γ⊢$_t$ {|´N=n|} ´R :== PROC Fac(´N) {|´R = fac n|}
apply (*hoare-rule HoareTotal.ProcRec1* [**where** *r*=*measure* (λ(*s*,*p*). ^{s}*N*)])
apply *vcg*
apply *simp*
done

procedures
p91(*R*,*N* | *R*) = *IF 100 < ´N THEN ´R :== ´N − 10*
ELSE ´R :== CALL p91(´R,´N+11);;
´R :== CALL p91(´R,´R) FI

p91-spec: ∀ *n*. Γ⊢$_t$ {|´N=n|} ´R :== PROC p91(´R,´N)
{|if 100 < n then ´R = n − 10 else ´R = 91|},{}

lemma (**in** *p91-impl*) *p91-spec*:
shows ∀ σ. Γ⊢$_t$ {σ} ´R :== PROC p91(´R,´N)
{|if 100 < ^{σ}N then ´R = ^{σ}N − 10 else ´R = 91|},{}
apply (*hoare-rule HoareTotal.ProcRec1* [**where** *r*=*measure* (λ(*s*,*p*). 101 − ^{s}*N*)])
apply *vcg*
apply *clarsimp*
apply *arith*
done

record *globals-list* =
next-′ :: *ref* ⇒ *ref*
cont-′ :: *ref* ⇒ *nat*

record ′*g list-vars* = ′*g state* +
p-′ :: *ref*
q-′ :: *ref*
r-′ :: *ref*
root-′ :: *ref*
tmp-′ :: *ref*

procedures
append(*p*,*q*|*p*) =
IF ´p=Null THEN ´p :== ´q ELSE ´p→´next :== CALL append(´p→´next,´
q) FI

lemma (**in** *append-impl*)
shows
∀ σ *Ps Qs*. Γ⊢$_t$

$\{\!|\sigma.\ List\ {}^\acute{}p\ {}^\acute{}next\ Ps \wedge\ \ List\ {}^\acute{}q\ {}^\acute{}next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}\ |\!\}$
$\quad {}^\acute{}p :== PROC\ append({}^\acute{}p,{}^\acute{}q)$
$\{\!|List\ {}^\acute{}p\ {}^\acute{}next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow {}^\acute{}next\ x = {}^\sigma next\ x)|\!\}$
apply (*hoare-rule HoareTotal.ProcRec1*
[**where** $r{=}measure\ (\lambda(s,p).\ length\ (list\ {}^sp\ {}^snext))$])
apply *vcg*
apply (*fastforce simp add*: *List-list*)
done

lemma (**in** *append-impl*)
shows
$\forall \sigma\ Ps\ Qs.\ \Gamma\vdash_t$
$\{\!|\sigma.\ List\ {}^\acute{}p\ {}^\acute{}next\ Ps \wedge\ \ List\ {}^\acute{}q\ {}^\acute{}next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}\ |\!\}$
$\quad {}^\acute{}p :== PROC\ append({}^\acute{}p,{}^\acute{}q)$
$\{\!|List\ {}^\acute{}p\ {}^\acute{}next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow {}^\acute{}next\ x = {}^\sigma next\ x)|\!\}$
apply (*hoare-rule HoareTotal.ProcRec1*
[**where** $r{=}measure\ (\lambda(s,p).\ length\ (list\ {}^sp\ {}^snext))$])
apply *vcg*
apply (*fastforce simp add*: *List-list*)
done

lemma (**in** *append-impl*)
shows
append-spec:
$\forall \sigma.\ \Gamma\vdash_t (\{\sigma\} \cap \{\!|islist\ {}^\acute{}p\ {}^\acute{}next|\!\})\ \ {}^\acute{}p :== PROC\ append({}^\acute{}p,{}^\acute{}q)$
$\{\!|\forall Ps\ Qs.\ List\ {}^\sigma p\ {}^\sigma next\ Ps \wedge\ \ List\ {}^\sigma q\ {}^\sigma next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}$
$\longrightarrow$
$List\ {}^\acute{}p\ {}^\acute{}next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow {}^\acute{}next\ x = {}^\sigma next\ x)|\!\}$
apply (*hoare-rule HoareTotal.ProcRec1*
[**where** $r{=}measure\ (\lambda(s,p).\ length\ (list\ {}^sp\ {}^snext))$])
apply *vcg*
apply *fastforce*
done

lemma $\Gamma\vdash\{\!|List\ {}^\acute{}p\ {}^\acute{}next\ Ps|\!\}$
${}^\acute{}q :== Null;;$
$WHILE\ {}^\acute{}p \neq Null\ INV\ \{\!|\exists Ps'\ Qs'.\ List\ {}^\acute{}p\ {}^\acute{}next\ Ps' \wedge List\ {}^\acute{}q\ {}^\acute{}next\ Qs' \wedge$
$\quad set\ Ps' \cap set\ Qs' = \{\} \wedge$
$\quad rev\ Ps'\ @\ Qs' = rev\ Ps|\!\}$
DO
${}^\acute{}r :== {}^\acute{}p;;\ {}^\acute{}p :== {}^\acute{}p \rightarrow {}^\acute{}next;;$
${}^\acute{}r \rightarrow {}^\acute{}next :== {}^\acute{}q;;\ {}^\acute{}q :== {}^\acute{}r$
$OD;;$
${}^\acute{}p :== {}^\acute{}q$
$\{\!|List\ {}^\acute{}p\ {}^\acute{}next\ (rev\ Ps)|\!\}$
apply *vcg*
apply *clarsimp*
apply *clarsimp*

apply *force*
apply *simp*
done

lemma *conjI2*: $[\![Q;\ Q \Longrightarrow P]\!] \Longrightarrow P \wedge Q$
by *blast*

procedures $Rev(p|p)$ =
 $\acute{q}$:== *Null*;;
 WHILE $\acute{p} \neq$ *Null*
 DO
  $\acute{r}$:== $\acute{p}$;; $\{\!|\acute{p} \neq Null|\!\}\longmapsto$ $\acute{p}$:== $\acute{p}\rightarrow\acute{next}$;;
  $\{\!|\acute{r} \neq Null|\!\}\longmapsto$ $\acute{r}\rightarrow\acute{next}$:== $\acute{q}$;; $\acute{q}$:== $\acute{r}$
 OD;;
 $\acute{p}$:==$\acute{q}$
Rev-spec:
$\forall Ps.\ \Gamma\vdash_t \{\!|List\ \acute{p}\ \acute{next}\ Ps|\!\}\ \acute{p}$:== *PROC* $Rev(\acute{p})$ $\{\!|List\ \acute{p}\ \acute{next}\ (rev\ Ps)|\!\}$
Rev-modifies:
$\forall \sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}\ \acute{p}$:== *PROC* $Rev(\acute{p})$ $\{t.\ t$ *may-only-modify-globals* σ *in* $[next]\}$

We only need partial correctness of modifies clause!

lemma *upd-hd-next*:
 assumes *p-ps*: *List p next* $(p\#ps)$
 shows *List* $(next\ p)$ $(next(p := q))$ *ps*
proof −
 from *p-ps*
 have $p \notin set\ ps$
  by *auto*
 with *p-ps* **show** *?thesis*
  by *simp*
qed

lemma (**in** *Rev-impl*) **shows**
Rev-spec:
$\forall Ps.\ \Gamma\vdash_t \{\!|List\ \acute{p}\ \acute{next}\ Ps|\!\}\ \acute{p}$:== *PROC* $Rev(\acute{p})$ $\{\!|List\ \acute{p}\ \acute{next}\ (rev\ Ps)|\!\}$
apply (*hoare-rule HoareTotal.ProcNoRec1*)
apply (*hoare-rule anno* =
 $\acute{q}$:== *Null*;;
 WHILE $\acute{p} \neq$ *Null INV* $\{\!|\exists Ps'\ Qs'.\ List\ \acute{p}\ \acute{next}\ Ps' \wedge List\ \acute{q}\ \acute{next}\ Qs' \wedge$
  $set\ Ps' \cap set\ Qs' = \{\} \wedge$
  $rev\ Ps'\ @\ Qs' = rev\ Ps|\!\}$
 VAR MEASURE (*length* (*list* $\acute{p}$ $\acute{next}$))
 DO
  $\acute{r}$:== $\acute{p}$;; $\{\!|\acute{p} \neq Null|\!\}\longmapsto\acute{p}$:== $\acute{p}\rightarrow\acute{next}$;;
  $\{\!|\acute{r} \neq Null|\!\}\longmapsto\acute{r}\rightarrow\acute{next}$:== $\acute{q}$;; $\acute{q}$:== $\acute{r}$
 OD;;
 $\acute{p}$:==$\acute{q}$ **in** *HoareTotal.annotateI*)
apply *vcg*

apply *clarsimp*
apply *clarsimp*
apply (*rule conjI2*)
apply *force*
apply *clarsimp*
apply (*subgoal-tac List p next* (*p#ps*))
prefer *2* **apply** *simp*
apply (*frule-tac q=q* **in** *upd-hd-next*)
apply (*simp only*: *List-list*)
apply *simp*
apply *simp*
done

lemma (**in** *Rev-impl*) **shows**
Rev-modifies:
$\forall \sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}$ ´*p* :== *PROC Rev*(´*p*) $\{t.$ *t may-only-modify-globals* σ *in* [*next*]$\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*vcg spec=modifies*)
done

lemma $\Gamma\vdash_t$ {|*List* ´*p* ´*next Ps*|}
´*q* :== *Null*;;
WHILE ´*p* $\neq$ *Null INV* {|$\exists$ *Ps' Qs'*. *List* ´*p* ´*next Ps'* $\wedge$ *List* ´*q* ´*next Qs'* $\wedge$
set Ps' $\cap$ *set Qs'* = {} $\wedge$
rev Ps' @ *Qs'* = *rev Ps*|}
VAR MEASURE (*length* (*list* ´*p* ´*next*))
DO
´*r* :== ´*p*;; ´*p* :== ´*p*→´*next*;;
´*r*→´*next* :== ´*q*;; ´*q* :== ´*r*
OD;;
´*p* :==´*q*
{|*List* ´*p* ´*next* (*rev Ps*)|}
apply *vcg*
apply *clarsimp*
apply *clarsimp*
apply (*rule conjI2*)
apply *force*
apply *clarsimp*
apply (*subgoal-tac List p next* (*p#ps*))
prefer *2* **apply** *simp*
apply (*frule-tac q=q* **in** *upd-hd-next*)
apply (*simp only*: *List-list*)
apply *simp*
apply *simp*
done

procedures
pedal(*N*,*M*) = *IF 0* < ´*N THEN*
IF 0 < ´*M THEN CALL coast*(´*N*− *1*,´*M*− *1*) *FI*;;
CALL pedal(´*N*− *1*,´*M*)
FI

and

coast(*N*,*M*) = *CALL pedal*(´*N*,´*M*);;
IF 0 < ´*M THEN CALL coast*(´*N*,´*M*− *1*) *FI*

ML ‹*ML-Thms.bind-thm* (*HoareTotal-ProcRec2*, *Hoare.gen-proc-rec* @{*context*} *Hoare.Total 2*)›

lemma (**in** *pedal-coast-clique*)
shows (Γ⊢$_t$ ⦃*True*⦄ *PROC pedal*(´*N*,´*M*) ⦃*True*⦄) ∧
(Γ⊢$_t$ ⦃*True*⦄ *PROC coast*(´*N*,´*M*) ⦃*True*⦄)
apply (*hoare-rule HoareTotal-ProcRec2*
[**where** *?r*= *inv-image* (*measure* (λ*m*. *m*) <∗*lex*∗>
measure (λ*p*. *if p* = *coast*-′*proc then 1 else 0*))
(λ(*s*,*p*). (^{s}*N* + ^{s}*M*,*p*))])
apply *simp-all*
apply *vcg*
apply *simp*
apply *vcg*
apply *simp*
done

lemma (**in** *pedal-coast-clique*)
shows (Γ⊢$_t$ ⦃*True*⦄ *PROC pedal*(´*N*,´*M*) ⦃*True*⦄) ∧
(Γ⊢$_t$ ⦃*True*⦄ *PROC coast*(´*N*,´*M*) ⦃*True*⦄)
apply (*hoare-rule HoareTotal-ProcRec2*
[**where** *?r*= *inv-image* (*measure* (λ*m*. *m*) <∗*lex*∗>
measure (λ*p*. *if p* = *coast*-′*proc then 1 else 0*))
(λ(*s*,*p*). (^{s}*N* + ^{s}*M*,*p*))])
apply *simp-all*
apply *vcg*
apply *simp*
apply *vcg*
apply *simp*
done

lemma (**in** *pedal-coast-clique*)
  shows ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC pedal*(´*N*,´*M*) $\{\!|True|\!\}$) $\wedge$
    ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC coast*(´*N*,´*M*) $\{\!|True|\!\}$)
  apply(*hoare-rule HoareTotal-ProcRec2*
    [**where** *?r*= *measure* (λ(*s*,*p*). ^sN + ^sM + (*if p* = *coast-'proc then 1 else 0*))])
  apply *simp-all*
  apply *vcg*
  apply *simp*
  apply *arith*
  apply *vcg*
  apply *simp*
  done

lemma (**in** *pedal-coast-clique*)
  shows ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC pedal*(´*N*,´*M*) $\{\!|True|\!\}$) $\wedge$
    ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC coast*(´*N*,´*M*) $\{\!|True|\!\}$)
  apply(*hoare-rule HoareTotal-ProcRec2*
    [**where** *?r*= (λ(*s*,*p*). ^sN) <**mlex**> (λ(*s*,*p*). ^sM) <**mlex**>
      measure (λ(*s*,*p*). *if p* = *coast-'proc then 1 else 0*)])
   apply *simp-all*
   apply *vcg*
   apply *simp*
   apply *vcg*
   apply *simp*
   done

lemma (**in** *pedal-coast-clique*)
  shows ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC pedal*(´*N*,´*M*) $\{\!|True|\!\}$) $\wedge$
    ($\Gamma\vdash_t$ $\{\!|True|\!\}$ *PROC coast*(´*N*,´*M*) $\{\!|True|\!\}$)
  apply(*hoare-rule HoareTotal-ProcRec2*
    [**where** *?r*= *measure* (λ*s*. ^sN + ^sM) <**lex**>
      measure (λ*p*. *if p* = *coast-'proc then 1 else 0*)])
   apply *simp-all*
   apply *vcg*
   apply *simp*
   apply *vcg*
   apply *simp*
   done

end

20 Example: Quicksort on Heap Lists

theory *Quicksort*
imports ../*Vcg* ../*HeapList HOL*−*Library.Permutation*
begin

record *globals-heap* =
  next-' :: *ref* ⇒ *ref*
  cont-' :: *ref* ⇒ *nat*

record *'g vars* = *'g state* +
  p-' :: *ref*
  q-' :: *ref*
  le-' :: *ref*
  gt-' :: *ref*
  hd-' :: *ref*
  tl-' :: *ref*

procedures
  append(p,q|p) =
    IF ´p=Null THEN ´p :== ´q ELSE ´p→´next :== CALL append(´p→´next,´q) FI

  append-spec:
  $\forall \sigma$ *Ps Qs*.
    $\Gamma\vdash$ {|σ. *List ´p ´next Ps* ∧ *List ´q ´next Qs* ∧ *set Ps* ∩ *set Qs* = {}|}
      ´p :== PROC append(´p,´q)
    {|*List ´p ´next* (*Ps*@*Qs*) ∧ (∀ *x*. *x*∉*set Ps* ⟶ *´next x* = $^{\sigma}$*next x*)|}

  append-modifies:
    $\forall \sigma$. $\Gamma\vdash$ {σ} *´p :== PROC append(´p,´q)*{*t*. *t may-only-modify-globals* σ *in* [*next*]}

lemma (**in** *append-impl*) *append-modifies*:
  shows
    $\forall \sigma$. $\Gamma\vdash$ {σ} *´p :== PROC append(´p,´q)*{*t*. *t may-only-modify-globals* σ *in* [*next*]}
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply (*vcg spec=modifies*)
  done

lemma (**in** *append-impl*) *append-spec*:
  shows $\forall \sigma$ *Ps Qs*. $\Gamma\vdash$
    {|σ. *List ´p ´next Ps* ∧ *List ´q ´next Qs* ∧ *set Ps* ∩ *set Qs* = {}|}
      ´p :== PROC append(´p,´q)
    {|*List ´p ´next* (*Ps*@*Qs*) ∧ (∀ *x*. *x*∉*set Ps* ⟶ *´next x* = $^{\sigma}$*next x*)|}
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *fastforce*
  done

primrec *sorted*:: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a list* ⇒ *bool*

```
where
sorted le [] = True |
sorted le (x#xs) = ((∀ y∈set xs. le x y) ∧ sorted le xs)

lemma perm-set-eq:
  assumes perm: xs <~~> ys
  shows set xs = set ys
  using perm
  by induct auto

lemma perm-Cons-eq [iff]: x#xs <~~> x#ys = (xs <~~> ys)
  by auto

lemma perm-app-Cons-eq1 : xs@y#ys <~~> zs = (y#xs@ys <~~> zs)
proof −
  have app-Cons: xs@y#ys <~~> y#xs@ys
    by (rule perm-sym, rule perm-append-Cons)
  show ?thesis
  proof
    assume xs@y#ys <~~> zs
    with app-Cons [THEN perm-sym]
    show y#xs@ys <~~> zs
      by (rule perm.trans)
  next
    assume  y#xs@ys <~~> zs
    with app-Cons
    show xs@y#ys <~~> zs
      by (rule perm.trans)
  qed
qed

lemma perm-app-Cons-eq2 : zs <~~> xs@y#ys = (zs <~~> y#xs@ys)
proof −
  have xs@y#ys <~~> zs = (y#xs@ys <~~> zs)
    by (rule perm-app-Cons-eq1)
  thus ?thesis
    by (iprover intro: perm-sym)
qed

lemmas perm-app-Cons-simps = perm-app-Cons-eq1 [THEN sym]
                             perm-app-Cons-eq2 [THEN sym]

lemma sorted-append[simp]:
 sorted le (xs@ys) = (sorted le xs ∧ sorted le ys ∧
                      (∀ x ∈ set xs. ∀ y ∈ set ys. le x y))
by (induct xs, auto)

lemma perm-append-blocks:
  assumes ws-ys: ws <~~> ys
```

assumes *xs-zs*: *xs* <~~> *zs*
shows *ws*@*xs* <~~> *ys*@*zs*
using *ws-ys*
proof (*induct*)
case (*swap l x y*)
from *xs-zs*
show (*l* # *x* # *y*) @ *xs* <~~> (*x* # *l* # *y*) @ *zs*
by (*induct*) *auto*
qed (*insert xs-zs* , *auto*)

procedures *quickSort*(*p*|*p*) =
IF ´p=Null THEN SKIP
ELSE ´tl :== ´p→´next;;
´le :== Null;;
´gt :== Null;;
WHILE ´tl≠Null DO
´hd :== ´tl;;
´tl :== ´tl→´next;;
IF ´hd→´cont ≤ ´p→´cont
THEN ´hd→´next :== ´le;;
´le :== ´hd
ELSE ´hd→´next :== ´gt;;
´gt :== ´hd
FI
OD;;
´le :== CALL quickSort(´le);;
´gt :== CALL quickSort(´gt);;
´p→´next :== ´gt;;
´le :== CALL append(´le,´p);;
´p :== ´le
FI

quickSort-spec:
$\forall \sigma$ *Ps*. Γ⊢ {|σ. *List ´p ´next Ps*|} *´p :== PROC quickSort*(*´p*)
{|(∃ *sortedPs*. *List ´p ´next sortedPs* ∧
sorted (≤) (*map* $^{\sigma}$*cont sortedPs*) ∧
Ps <~~> *sortedPs*) ∧
(∀ *x*. *x*∉*set Ps* ⟶ *´next x* = $^{\sigma}$*next x*)|}

quickSort-modifies:
∀ σ. Γ⊢ {σ} *´p :== PROC quickSort*(*´p*) {*t*. *t may-only-modify-globals* σ *in* [*next*]}

lemma (**in** *quickSort-impl*) *quickSort-modifies*:
shows
∀ σ. Γ⊢ {σ} *´p :== PROC quickSort*(*´p*) {*t*. *t may-only-modify-globals* σ *in* [*next*]}
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

```
lemma (in quickSort-impl) quickSort-spec:
shows
  ∀σ Ps. Γ⊢ {|σ. List ´p ´next Ps|}
              ´p :== PROC quickSort(´p)
             {|(∃sortedPs. List ´p ´next sortedPs ∧
              sorted (≤) (map σcont sortedPs) ∧
              Ps <~~> sortedPs) ∧
              (∀x. x∉set Ps ⟶ ´next x = σnext x)|}
apply (hoare-rule HoarePartial.ProcRec1)
apply (hoare-rule anno =
 IF ´p=Null THEN SKIP
  ELSE ´tl :== ´p→´next;;
       ´le :== Null;;
       ´gt :== Null;;
       WHILE ´tl≠Null
       INV {| (∃les grs tls. List ´le ´next les ∧ List ´gt ´next grs ∧
             List ´tl ´next tls ∧
             Ps <~~> ´p#tls@les@grs ∧
             distinct(´p#tls@les@grs) ∧
             (∀x∈set les. x→´cont ≤ ´p→´cont) ∧
             (∀x∈set grs. ´p→´cont < x→´cont)) ∧
             ´p=σp ∧
             ´cont=σcont ∧
             List σp σnext Ps ∧
             (∀x. x∉set Ps ⟶ ´next x = σnext x)|}
       DO
         ´hd :== ´tl;;
         ´tl :== ´tl→´next;;
         IF ´hd→´cont ≤ ´p→´cont
         THEN ´hd→´next :== ´le;;
              ´le :== ´hd
         ELSE ´hd→´next :== ´gt;;
              ´gt :== ´hd
         FI
       OD;;
       ´le :== CALL quickSort(´le);;
       ´gt :== CALL quickSort(´gt);;
       ´p→´next :== ´gt;;
       ´le :== CALL append(´le,´p);;
       ´p :== ´le
  FI in HoarePartial.annotateI)
apply vcg
apply  fastforce
apply  clarsimp
apply  (rule conjI)
apply   clarify
apply   (rule conjI)
apply    (rule-tac x=tl#les in exI)
```

apply *simp*
apply (*rule-tac x=grs* **in** *exI*)
apply *simp*
apply (*rule-tac x=ps* **in** *exI*)
apply *simp*
apply (*erule perm.trans*)
apply *simp*
apply (*simp add*: *perm-app-Cons-simps*)
apply (*simp add*: *perm-set-eq*)
apply *clarify*
apply (*rule conjI*)
apply (*rule-tac x=les* **in** *exI*)
apply *simp*
apply (*rule-tac x=tl#grs* **in** *exI*)
apply *simp*
apply (*rule-tac x=ps* **in** *exI*)
apply *simp*
apply (*erule perm.trans*)
apply *simp*
apply (*simp add*: *perm-app-Cons-simps*)
apply (*simp add*: *perm-set-eq*)
apply *clarsimp*
apply (*rule-tac ?x=grs* **in** *exI*)
apply (*rule conjI*)
apply (*erule heap-eq-ListI1*)
apply *clarify*
apply (*erule-tac x=x* **in** *allE*)**back**
apply *blast*
apply *clarsimp*
apply (*rule-tac x=sortedPs* **in** *exI*)
apply (*rule conjI*)
apply (*erule heap-eq-ListI1*)
apply (*clarsimp*)
apply (*erule-tac x=x* **in** *allE*) **back back**
apply (*fast dest*!: *perm-set-eq*)
apply (*rule-tac x=p#sortedPsa* **in** *exI*)
apply (*rule conjI*)
apply (*fastforce dest*!: *perm-set-eq*)
apply (*rule conjI*)
apply (*force dest*!: *perm-set-eq*)
apply *clarsimp*
apply (*rule conjI*)
apply (*fastforce dest*!: *perm-set-eq*)
apply (*rule conjI*)
apply (*fastforce dest*!: *perm-set-eq*)
apply (*rule conjI*)
apply (*erule perm.trans*)
apply (*simp add*: *perm-app-Cons-simps list-all-iff*)
apply (*fastforce intro*!: *perm-append-blocks*)

apply *clarsimp*
apply (*erule-tac x=x* **in** *allE*)+
apply (*force dest*!: *perm-set-eq*)
done

end

theory *XVcg*
imports *Vcg*

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

21 Examples for Parallel Assignments

theory *XVcgEx*
imports ../*XVcg*

begin

record *globals* =
  G-′::*nat*
  H-′::*nat*

```
record 'g vars = 'g state +
  A-' :: nat
  B-' :: nat
  C-' :: nat
  I-' :: nat
  M-' :: nat
  N-' :: nat
  R-' :: nat
  S-' :: nat
  Arr-' :: nat list
  Abr-':: string

term BASIC
        ´A :== x,
        ´B :== y
     END

term BASIC
        ´G :== ´H,
        ´H :== ´G
     END

term BASIC
       LET (x,y) = (´A,b);
           z = ´B
       IN ´A :== x,
          ´G :== ´A + y + z
     END


lemma Γ⊢ {|´A = 0|}
     {|´A < 0|} ⟼ BASIC
      LET (a,b,c) = foo ´A
      IN
           ´A :== a,
           ´B :== b,
           ´C :== c
     END
     {|´A = x ∧ ´B = y ∧ ´C = c|}
apply vcg
oops

lemma Γ⊢ {|´A = 0|}
     {|´A < 0|} ⟼ BASIC
      LET (a,b,c) = foo ´A
      IN
           ´A :== a,
           ´G :== b + ´B,
           ´H :== c
```

END
$\{\!|\, ´A = x \wedge ´G = y \wedge ´H = c |\!\}$
apply *vcg*
oops

definition *foo*:: $nat \Rightarrow (nat \times nat \times nat)$
where $foo\ n = (n,n+1,n+2)$

lemma $\Gamma\vdash \{\!|\, ´A = 0 |\!\}$
$\{\!|\, ´A < 0 |\!\} \longmapsto BASIC$
$LET\ (a,b,c) = foo\ ´A$
IN
$´A :== a,$
$´G :== b + ´B,$
$´H :== c$
END
$\{\!|\, ´A = x \wedge ´G = y \wedge ´H = c |\!\}$
apply (*vcg add*: *foo-def snd-conv fst-conv*)
oops

end

22 Examples for Procedures as Parameters

theory *ProcParEx* **imports** ../*Vcg* **begin**

lemma *DynProcProcPar'*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge$
$(\exists Z.\ init\ s \in P'\ Z \wedge$
$(\forall t \in Q'\ Z.\ return\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ Z.\ return\ s\ t \in A))\}$
assumes *result*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ result\ s\ t\ Q,A$
assumes *q*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ Call\ q\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F} P\ dynCall\ init\ p\ return\ result\ Q,A$
apply (*rule HoarePartial.DynProcProcPar* [*OF* - *result q*])
apply (*insert adapt*)
apply *fast*
done

lemma *conseq-exploit-pre'*:
$[\![\forall s \in S.\ \Gamma,\Theta \vdash (\{s\} \cap P)\ c\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash (P \cap S)c\ Q,A$
apply (*rule HoarePartialDef.Conseq*)
apply *clarify*
by (*metis IntI insertI1 subset-refl*)

lemma *conseq-exploit-pre''*:
$[\![\forall Z. \forall s \in S\ Z.\ \ \Gamma,\Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z),(A\ Z)]\!]$
$\Longrightarrow$
$\forall Z. \Gamma,\Theta\vdash (P\ Z \cap S\ Z) c\ (Q\ Z),(A\ Z)$
apply (*rule allI*)
apply (*rule conseq-exploit-pre'*)
apply *blast*
done

lemma *conseq-exploit-pre'''*:
$[\![\forall s \in S. \forall Z. \Gamma,\Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z),(A\ Z)]\!]$
$\Longrightarrow$
$\forall Z. \Gamma,\Theta\vdash (P\ Z \cap S) c\ (Q\ Z),(A\ Z)$
apply (*rule allI*)
apply (*rule conseq-exploit-pre'*)
apply *blast*
done

record *'g vars = 'g state +*
compare-' :: *string*
n-' :: *nat*
m-' :: *nat*
b-' :: *bool*
k-' :: *nat*

procedures *compare*(*n*,*m*|*b*) = *NoBody*
print-locale! *compare-signature*

context *compare-signature*
begin
declare [[*hoare-use-call-tr'* = *false*]]
term *´b* :== *CALL compare*(*´n*,*´m*)
term *´b* :== *DYNCALL ´compare*(*´n*,*´m*)
declare [[*hoare-use-call-tr'* = *true*]]
term *´b* :== *DYNCALL ´compare*(*´n*,*´m*)
end

procedures
LEQ (*n*,*m* | *b*) = *´b* :== *´n* $\leq$ *´m*
LEQ-spec: $\forall \sigma$. $\Gamma\vdash$ $\{\sigma\}$ *PROC LEQ*(*´n*,*´m*,*´b*) $\{\!|$ *´b* = ($^{\sigma}n \leq {}^{\sigma}m$)$|\!\}$
LEQ-modifies: $\forall \sigma$. $\Gamma\vdash$ $\{\sigma\}$ *PROC LEQ*(*´n*,*´m*,*´b*) $\{t.$ *t may-only-modify-globals*

σ *in* []}

definition mx:: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
  where $mx\ leq\ a\ b = (if\ leq\ a\ b\ then\ a\ else\ b)$

procedures
  Max (*compare*, *n*, *m* | *k*) =
  ´*b* :== *DYNCALL* ´*compare*(´*n*,´*m*);;
   IF ´*b* *THEN* ´*k* :== ´*n* *ELSE* ´*k* :== ´*m* *FI*

  Max-spec: $\bigwedge leq.\ \forall\sigma.\ \Gamma\vdash$
  $(\{\sigma\} \cap \{s.\ (\forall\tau.\ \Gamma\vdash\ \{\tau\}$ ´*b* :== *PROC* ^{s}*compare*(´*n*,´*m*) ⦃´*b* = (*leq* $^\tau n$ $^\tau m$)⦄) ∧
          $(\forall\tau.\ \Gamma\vdash\ \{\tau\}$ ´*b* :== *PROC* ^{s}*compare*(´*n*,´*m*) {*t*. *t may-only-modify-globals*
τ *in* []})})
    PROC Max(´*compare*,´*n*,´*m*,´*k*)
  ⦃´*k* = *mx leq* $^\sigma n$ $^\sigma m$⦄

lemma (**in** *Max-impl*) *Max-spec1*:
shows
$\forall\sigma\ leq.\ \Gamma\vdash$
  $(\{\sigma\} \cap$ ⦃ $(\forall\tau.\ \Gamma\vdash\{\tau\}$ ´*b* :== *PROC* ´*compare*(´*n*,´*m*) ⦃´*b* = (*leq* $^\tau n$ $^\tau m$)⦄) ∧
     $(\forall\tau.\ \Gamma\vdash\ \{\tau\}$ ´*b* :== *PROC* ´*compare*(´*n*,´*m*) {*t*. *t may-only-modify-globals* τ
in []})⦄)
    ´*k* :== *PROC Max*(´*compare*,´*n*,´*m*)
  ⦃´*k* = *mx leq* $^\sigma n$ $^\sigma m$⦄
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre*′)
apply (*rule*)
apply *clarify*
proof −
  fix σ:: ('*a*,'*b*) *vars-scheme* **and** *s*::('*a*,'*b*) *vars-scheme* **and** *leq*
   assume *compare-spec*:
     $\forall\tau.\ \Gamma\vdash\{\tau\}$ ´*b* :== *PROC* ^{s}*compare*(´*n*,´*m*) ⦃´*b* = *leq* $^\tau n$ $^\tau m$⦄

  assume *compare-modifies*:
     $\forall\tau.\ \Gamma\vdash\{\tau\}$ ´*b* :== *PROC* ^{s}*compare*(´*n*,´*m*)
          {*t*. *t may-only-modify-globals* τ *in* []}

   show $\Gamma\vdash(\{s\} \cap \{\sigma\})$
          ´*b* :== *DYNCALL* ´*compare* (´*n*,´*m*);;
          IF ´*b* *THEN* ´*k* :== ´*n* *ELSE* ´*k* :== ´*m* *FI*
          ⦃´*k* = *mx leq* $^\sigma n$ $^\sigma m$⦄
    apply *vcg*
    apply (*clarsimp simp add*: *mx-def*)
    done

qed

lemma (**in** *Max-impl*) *Max-spec2*:
shows
$\forall\,\sigma\ leq.\ \Gamma\vdash$
$(\{\sigma\} \cap \{\!|(\forall\,\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}b :== PROC\ \acute{}compare(\acute{}n,\acute{}m)\ \{\!|\acute{}b = (leq\ {}^{\tau}n\ {}^{\tau}m)|\!\}) \wedge$
$(\forall\,\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}b :== PROC\ \acute{}compare(\acute{}n,\acute{}m)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau$
$in\ []\})|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *Max-impl*) *Max-spec3*:
shows
$\forall\,n\ m\ leq.\ \Gamma\vdash$
$(\{\!|\acute{}n=n \wedge \acute{}m=m|\!\} \cap$
$\{\!|(\forall\,\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}b :== PROC\ \acute{}compare(\acute{}n,\acute{}m)\ \{\!|\acute{}b = (leq\ {}^{\tau}n\ {}^{\tau}m)|\!\}) \wedge$
$(\forall\,\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}b :== PROC\ \acute{}compare(\acute{}n,\acute{}m)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau$
$in\ []\})|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ n\ m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *Max-impl*) *Max-spec4*:
shows
$\forall\,n\ m\ leq.\ \Gamma\vdash$
$(\{\!|\acute{}n=n \wedge \acute{}m=m|\!\} \cap \{\!|\forall\,\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}b :== PROC\ \acute{}compare(\acute{}n,\acute{}m)\ \{\!|\acute{}b = (leq\ {}^{\tau}n$
${}^{\tau}m)|\!\}|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ n\ m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)

apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

locale *Max-test* = *Max-spec* + *LEQ-spec* + *LEQ-modifies*
lemma (**in** *Max-test*)

shows
$\Gamma\vdash \{\sigma\}$ ´*k* :== *CALL Max*(*LEQ-′proc*,´*n*,´*m*) $\{\!|$´$k = mx\ (\leq)\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$
proof −
note *Max-spec* = *Max-spec* [**where** *leq*=(≤)]
show *?thesis*
apply *vcg*
apply (*clarsimp*)
apply (*rule conjI*)
apply (*rule LEQ-spec* [*simplified*])
apply (*rule LEQ-modifies* [*simplified*])
done
qed

lemma (**in** *Max-impl*) *Max-spec5*:
shows
$\forall\, n\ m\ leq.\ \Gamma\vdash$
($\{\!|$´n=n ∧ ´m=$m|\!\}$ ∩ $\{\!|\forall\, n'\ m'.\ \Gamma\vdash \{\!|$´$n$=$n'$ ∧ ´m=$m'|\!\}$ ´*b* :== *PROC* ´*compare*(´
n,´*m*) $\{\!|$´$b = (leq\ n'\ m')|\!\}|\!\}$)
´*k* :== *PROC Max*(´*compare*,´*n*,´*m*)
$\{\!|$´$k = mx\ leq\ n\ m|\!\}$
term $\{\!|\{s.\ {}^{s}n = n' \wedge {}^{s}m = m'\} = X|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre′*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply *clarsimp*
apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *LEQ-impl*)
LEQ-spec: $\forall\, n\ m.\ \Gamma\vdash \{\!|$´$n$=$n$ ∧ ´m=$m|\!\}$ *PROC LEQ*(´*n*,´*m*,´*b*) $\{\!|$´$b = (n \leq m)|\!\}$
apply *vcg*
done

locale *Max-test′* = *Max-impl* + *LEQ-impl*
lemma (**in** *Max-test′*)
shows

$\forall n\ m.\ \Gamma\vdash \{\!|\,\acute{}n{=}n \wedge \acute{}m{=}m|\!\}\ \acute{}k :== CALL\ Max(LEQ\text{-}'proc,\acute{}n,\acute{}m)\ \{\!|\,\acute{}k = mx\ (\leq)\ n\ m|\!\}$
proof –
  note *Max-spec* = *Max-spec5*
  show *?thesis*
    apply *vcg*
    apply (*rule-tac x*=($\leq$) **in** *exI*)
    apply *clarsimp*
    apply (*rule LEQ-spec* [*rule-format*])
    done
qed

end

23 Examples for Procedures as Parameters using Statespaces

theory *ProcParExSP* **imports** ../*Vcg* **begin**

lemma *DynProcProcPar'*:
 assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge$
 $(\exists Z.\ init\ s \in P'\ Z \wedge$
 $(\forall t \in Q'\ Z.\ return\ s\ t \in R\ s\ t) \wedge$
 $(\forall t \in A'\ Z.\ return\ s\ t \in A))\}$
 assumes *result*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ result\ s\ t\ Q,A$
 assumes *q*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ q\ (Q'\ Z),(A'\ Z)$
 shows $\Gamma,\Theta\vdash_{/F}\ P\ dynCall\ init\ p\ return\ result\ Q,A$
apply (*rule HoarePartial.DynProcProcPar* [*OF - result q*])
apply (*insert adapt*)
apply *fast*
done

lemma *conseq-exploit-pre'*:
 $[\![\forall s \in S.\ \Gamma,\Theta \vdash (\{s\} \cap P)\ c\ Q,A]\!]$
 $\Longrightarrow$
 $\Gamma,\Theta\vdash\ (P \cap S)c\ Q,A$
  apply (*rule HoarePartialDef.Conseq*)
  apply *clarify*
  by (*metis IntI insertI1 subset-refl*)

lemma *conseq-exploit-pre''*:
 $[\![\forall Z.\ \forall s \in S\ Z.\ \ \Gamma,\Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z),(A\ Z)]\!]$
 $\Longrightarrow$
 $\forall Z.\ \Gamma,\Theta\vdash\ (P\ Z \cap S\ Z)c\ (Q\ Z),(A\ Z)$

apply (*rule allI*)
apply (*rule conseq-exploit-pre'*)
apply *blast*
done

lemma *conseq-exploit-pre'''*:
$\llbracket \forall s \in S. \forall Z. \Gamma,\Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z),(A\ Z) \rrbracket$
$\Longrightarrow$
$\forall Z. \Gamma,\Theta \vdash (P\ Z \cap S) c\ (Q\ Z),(A\ Z)$
apply (*rule allI*)
apply (*rule conseq-exploit-pre'*)
apply *blast*
done

procedures *compare*(*i*::*nat*,*j*::*nat*|*r*::*bool*) *NoBody*

print-locale! *compare-signature*

context *compare-impl*
begin
declare [[*hoare-use-call-tr'* = *false*]]
term *´r* :== *CALL compare*(*´i*, *´j*)
declare [[*hoare-use-call-tr'* = *true*]]
end

procedures
LEQ (*i*::*nat*,*j*::*nat* | *r*::*bool*) *´r* :== *´i* ≤ *´j*
LEQ-spec: $\forall \sigma. \Gamma \vdash \{\sigma\}$ *PROC LEQ*(*´i*,*´j*,*´r*) $\{\!|$ *´r* = ($^\sigma i \le {}^\sigma j$)$|\!\}$

LEQ-modifies: $\forall \sigma. \Gamma \vdash \{\sigma\}$ *PROC LEQ*(*´i*,*´j*,*´r*) {*t*. *t may-only-modify-globals* σ *in* []}

definition *mx*:: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a* ⇒ *'a* ⇒ *'a*
where *mx leq a b* = (*if leq a b then a else b*)

procedures (**imports** *compare-signature*)
Max (*compare*::*string*, *n*::*nat*, *m*::*nat* | *k*::*nat*)
where *b*::*bool*
in
´b :== *DYNCALL ´compare*(*´n*,*´m*);;
IF ´b THEN ´k :== *´n ELSE ´k* :== *´m FI*

Max-spec: $\bigwedge leq.\ \forall\sigma.\ \Gamma\vdash$
$(\{\sigma\} \cap \{s.\ (\forall\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}r :== PROC\ {}^{s}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = (leq\ {}^{\tau}i\ {}^{\tau}j)|\!\}) \wedge$
$(\forall\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}r :== PROC\ {}^{s}compare(\acute{}i,\acute{}j)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals$
$\tau\ in\ []\})\})$
$PROC\ Max(\acute{}compare,\acute{}n,\acute{}m,\acute{}k)$
$\{\!|\acute{}k = mx\ leq\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$

context *Max-spec*
begin
thm *Max-spec*
end
context *Max-impl*
begin
term $\acute{}b :== DYNCALL\ \acute{}compare(\acute{}n,\acute{}m)$
declare [[*hoare-use-call-tr′* = *false*]]
term $\acute{}b :== DYNCALL\ \acute{}compare(\acute{}n,\acute{}m)$
declare [[*hoare-use-call-tr′* = *true*]]
end

lemma (**in** *Max-impl*) *Max-spec1*:
shows
$\forall\sigma\ leq.\ \Gamma\vdash$
$(\{\sigma\} \cap \{\!|\ (\forall\tau.\ \Gamma\vdash\{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = (leq\ {}^{\tau}i\ {}^{\tau}j)|\!\}) \wedge$
$(\forall\tau.\ \Gamma\vdash\ \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau\ in$
$[]\})|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre′*)
apply (*rule*)
apply *clarify*
proof −
fix σ:: ($'a$, $'b$, $'c$, $'d$) *stateSP-scheme* **and** s::($'a$, $'b$, $'c$, $'d$) *stateSP-scheme* **and** *leq*
assume *compare-spec*:
$\forall\tau.\ \Gamma\vdash\{\tau\}\ \acute{}r :== PROC\ {}^{s}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = leq\ {}^{\tau}i\ {}^{\tau}j|\!\}$

assume *compare-modifies*:
$\forall\tau.\ \Gamma\vdash\{\tau\}\ \acute{}r :== PROC\ {}^{s}compare(\acute{}i,\acute{}j)$
$\{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau\ in\ []\}$

show $\Gamma\vdash(\{s\} \cap \{\sigma\})$
$\acute{}b :== DYNCALL\ \acute{}compare\ (\acute{}n,\acute{}m);;$
$IF\ \acute{}b\ THEN\ \acute{}k :== \acute{}n\ ELSE\ \acute{}k :== \acute{}m\ FI$
$\{\!|\acute{}k = mx\ leq\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$
apply *vcg*

apply (*clarsimp simp add*: *mx-def*)
done
qed

lemma (**in** *Max-impl*) *Max-spec2*:
shows
$\forall\sigma$ *leq*. $\Gamma\vdash$
$(\{\sigma\} \cap \{\!|(\forall\tau.\ \Gamma\vdash \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = (leq\ {}^{\tau}i\ {}^{\tau}j)|\!\}) \wedge$
$(\forall\tau.\ \Gamma\vdash \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau\ in\ []\})|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ {}^{\sigma}n\ {}^{\sigma}m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *Max-impl*) *Max-spec3*:
shows
$\forall n\ m\ leq.\ \Gamma\vdash$
$(\{\!|\acute{}n=n \wedge \acute{}m=m|\!\} \cap$
$\{\!|(\forall\tau.\ \Gamma\vdash \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = (leq\ {}^{\tau}i\ {}^{\tau}j)|\!\}) \wedge$
$(\forall\tau.\ \Gamma\vdash \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \tau\ in\ []\})|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ n\ m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *Max-impl*) *Max-spec4*:
shows
$\forall n\ m\ leq.\ \Gamma\vdash$
$(\{\!|\acute{}n=n \wedge \acute{}m=m|\!\} \cap \{\!|\forall\tau.\ \Gamma\vdash \{\tau\}\ \acute{}r :== PROC\ \acute{}compare(\acute{}i,\acute{}j)\ \{\!|\acute{}r = (leq\ {}^{\tau}i\ {}^{\tau}j)|\!\}|\!\})$
$\acute{}k :== PROC\ Max(\acute{}compare,\acute{}n,\acute{}m)$
$\{\!|\acute{}k = mx\ leq\ n\ m|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)

apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply (*clarsimp simp add*: *mx-def*)
done

print-locale *Max-spec*

locale *Max-test* = *Max-spec* **where**
  i-'compare-' = *i-'LEQ-'* **and**
  j-'compare-' = *j-'LEQ-'* **and**
  r-'compare-' = *r-'LEQ-'*
  + *LEQ-spec* + *LEQ-modifies*

lemma (**in** *Max-test*)
  shows
  $\Gamma\vdash \{\sigma\}$ ´*k* :== *CALL Max*(*LEQ-'proc*,´*n*,´*m*) $\{\!|$´*k* = *mx* $(\leq)$ $^\sigma n$ $^\sigma m|\!\}$
proof −
  note *Max-spec* = *Max-spec* [**where** *leq*=$(\leq)$]
  show *?thesis*
    apply *vcg*
    apply (*clarsimp*)
    apply (*rule conjI*)
    apply (*rule LEQ-spec*)
    apply (*rule LEQ-modifies*)
    done
qed

lemma (**in** *Max-impl*) *Max-spec5*:
shows
$\forall$ *n m leq*. $\Gamma\vdash$
  ($\{\!|$´*n*=*n* $\wedge$ ´*m*=*m*$|\!\}$ $\cap$ $\{\!|\forall$ *n' m'*. $\Gamma\vdash$ $\{\!|$´*i*=*n'* $\wedge$ ´*j*=*m'*$|\!\}$ ´*r* :== *PROC* ´*compare*(´
i,´*j*) $\{\!|$´*r* = (*leq n' m'*)$|\!\}|\!\}$)
   ´*k* :== *PROC Max*(´*compare*,´*n*,´*m*)
  $\{\!|$´*k* = *mx leq n m*$|\!\}$
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*intro allI*)
apply (*rule conseq-exploit-pre'*)
apply (*rule*)
apply *clarify*
apply *vcg*
apply *clarsimp*

apply (*clarsimp simp add*: *mx-def*)
done

lemma (**in** *LEQ-impl*)
LEQ-spec: $\forall n\ m.\ \Gamma\vdash \{\!|\acute{}i{=}n \wedge \acute{}j{=}m|\!\}\ \ PROC\ LEQ(\acute{}i,\acute{}j,\acute{}r)\ \{\!|\acute{}r = (n \leq m)|\!\}$
apply *vcg*
apply *simp*
done

print-locale *Max-impl*
locale *Max-test'* = *Max-impl* **where**
i-'compare-' = *i-'LEQ-'* **and**
j-'compare-' = *j-'LEQ-'* **and**
r-'compare-' = *r-'LEQ-'*
\+ *LEQ-impl*
lemma (**in** *Max-test'*)
shows
$\forall n\ m.\ \Gamma\vdash \{\!|\acute{}n{=}n \wedge \acute{}m{=}m|\!\}\ \acute{}k :== CALL\ Max(LEQ\text{-}'proc,\acute{}n,\acute{}m)\ \{\!|\acute{}k = mx\ (\leq)$
$n\ m|\!\}$
proof −
note *Max-spec* = *Max-spec5*
show *?thesis*
apply *vcg*
apply (*rule-tac x*=(≤) **in** *exI*)
apply *clarsimp*
apply (*rule LEQ-spec* [*rule-format*])
done
qed

end

24 Experiments with Closures

theory *Closure*
imports *../Hoare*
begin

definition
callClosure upd cl = *Seq* (*Basic* (*upd* (*fst cl*))) (*Call* (*snd cl*))

definition
dynCallClosure init upd cl return c =
DynCom ($\lambda s.$ *call* (*upd* (*fst* (*cl s*)) ∘ *init*) (*snd* (*cl s*)) *return c*)

lemma *dynCallClosure-sound*:
assumes *adapt*:
  $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P'\ (callClosure\ upd\ (cl\ s))\ Q',A' \wedge$
$\qquad init\ s \in P' \wedge$
$\qquad (\forall t \in Q'.\ return\ s\ t \in R\ s\ t) \wedge$
$\qquad (\forall t \in A'.\ return\ s\ t \in A)\}$
assumes *res*: $\forall s\ t\ n.\ \Gamma,\Theta \models n{:}_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
shows
$\Gamma,\Theta \models n{:}_{/F}\ P\ (dynCallClosure\ init\ upd\ cl\ return\ c)\ Q,A$
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*: $\Gamma \vdash \langle dynCallClosure\ init\ upd\ cl\ return\ c, Normal\ s\rangle = n \Rightarrow t$
  from *execn.Basic* [**where** $f=(upd\ (fst\ (cl\ s)))$ **and** $s=(init\ s)$]
  have *exec-upd*: $\Gamma \vdash \langle Basic\ (upd\ (fst\ (cl\ s))), Normal\ (init\ s)\rangle = n \Rightarrow$
$\qquad Normal\ (((upd\ (fst\ (cl\ s))) \circ init)\ s)$
    by *auto*
  assume *P*: $s \in P$
  from *P adapt* **obtain** $P'\ Q'\ A'$
    where
    valid: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P'\ (callClosure\ upd\ (cl\ s))\ Q',A'$ **and**
    init-P': $init\ s \in P'$ **and**
    R: $(\forall t \in Q'.\ return\ s\ t \in R\ s\ t)$ **and**
    A: $(\forall t \in A'.\ return\ s\ t \in A)$
    by *auto*
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  from *exec* [*simplified dynCallClosure-def*]
  have *exec-call*:
    $\Gamma \vdash \langle call\ (upd\ (fst\ (cl\ s)) \circ init)\ (snd\ (cl\ s))\ return\ c, Normal\ s\rangle = n \Rightarrow t$
   by *cases*
  then
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases rule*: *execn-call-Normal-elim*)
    fix *bdy m t'*
    assume *bdy*: $\Gamma\ (snd\ (cl\ s)) = Some\ bdy$
    assume *exec-body*: $\Gamma \vdash \langle bdy, Normal\ ((upd\ (fst\ (cl\ s)) \circ init)\ s)\rangle = m \Rightarrow Normal$
t'
    assume *exec-c*: $\Gamma \vdash \langle c\ s\ t', Normal\ (return\ s\ t')\rangle = Suc\ m \Rightarrow t$
    assume *n*: $n = Suc\ m$
    have $\Gamma \vdash \langle Basic\ init, Normal\ s\rangle = m \Rightarrow Normal\ (init\ s)$
      by (*rule execn.Basic*)
    from *bdy exec-body*
    have *exec-callC*:
      $\Gamma \vdash \langle Call\ (snd\ (cl\ s)), Normal\ ((upd\ (fst\ (cl\ s)) \circ init)\ s)\rangle = Suc\ m \Rightarrow Normal$
t'
      by (*rule execn.Call*)

from *execn.Seq* [*OF exec-upd* [*simplified n*]*exec-callC*]
have *exec-closure*: $\Gamma\vdash$ $\langle$*callClosure upd* (*cl s*),*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Normal t'*
by (*simp add*: *callClosure-def n*)
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-closure init-P'*]
have *t'* $\in$ *Q'*
by *auto*
with *R* **have** *return s t'* $\in$ *R s t'*
by *auto*
from *cnvalidD* [*OF res* [*rule-format*] *ctxt exec-c* [*simplified n*[*symmetric*]] *this t-notin-F*]
show *?thesis*
by *auto*
next
fix *bdy m t'*
assume *bdy*: Γ (*snd* (*cl s*)) = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*m*$\Rightarrow$ *Abrupt t'*
assume *t*: *t*=*Abrupt* (*return s t'*)
assume *n*: *n* = *Suc m*
from *bdy exec-body*
have *exec-callC*:
$\Gamma\vdash\langle$*Call* (*snd* (*cl s*)),*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*Suc m*$\Rightarrow$ *Abrupt t'*
by (*rule execn.Call*)
from *execn.Seq* [*OF exec-upd* [*simplified n*] *exec-callC*]
have *exec-closure*: $\Gamma\vdash$ $\langle$*callClosure upd* (*cl s*),*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Abrupt t'*
by (*simp add*: *callClosure-def n*)

from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-closure init-P'*]
have *t'* $\in$ *A'*
by *auto*
with *A* **have** *return s t'* $\in$ *A*
by *auto*
with *t* **show** *?thesis*
by *auto*
next
fix *bdy m f*
assume *bdy*: Γ (*snd* (*cl s*)) = *Some bdy*
assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*m*$\Rightarrow$ *Fault f*
assume *t*: *t*=*Fault f*
assume *n*: *n* = *Suc m*
from *bdy exec-body*
have *exec-callC*:
$\Gamma\vdash\langle$*Call* (*snd* (*cl s*)),*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*Suc m*$\Rightarrow$ *Fault f*
by (*rule execn.Call*)
from *execn.Seq* [*OF exec-upd* [*simplified n*] *exec-callC*]
have *exec-closure*: $\Gamma\vdash$ $\langle$*callClosure upd* (*cl s*),*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Fault f*
by (*simp add*: *callClosure-def n*)
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-closure init-P'*] *t-notin-F t*
have *False*

    by *auto*
  thus *?thesis* **..**
next
  fix *bdy m*
  assume *bdy*: Γ (*snd* (*cl s*)) = *Some bdy*
  assume *exec-body*: $\Gamma\vdash\langle$*bdy*,*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*m*$\Rightarrow$ *Stuck*
  assume *t*: *t*=*Stuck*
  assume *n*: *n* = *Suc m*
  from *execn.Basic* [**where** *f*=(*upd* (*fst* (*cl s*))) **and** *s*=(*init s*)]
  have *exec-upd*: $\Gamma\vdash\langle$*Basic* (*upd* (*fst* (*cl s*))),*Normal* (*init s*)$\rangle$ =*Suc m*$\Rightarrow$
        Normal (((*upd* (*fst* (*cl s*))) $\circ$ *init*) *s*)
    by *auto*
  from *bdy exec-body*
  have *exec-callC*:
    $\Gamma\vdash\langle$*Call* (*snd* (*cl s*)),*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*Suc m*$\Rightarrow$ *Stuck*
    by (*rule execn.Call*)
  from *execn.Seq* [*OF exec-upd* [*simplified n*] *exec-callC*]
  have *exec-closure*: $\Gamma\vdash$ $\langle$*callClosure upd* (*cl s*),*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Stuck*
    by (*simp add*: *callClosure-def n*)
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-closure init-P'*] *t-notin-F t*
  have *False*
    by *auto*
  thus *?thesis* **..**
next
  fix *m*
  assume *no-bdy*: Γ (*snd* (*cl s*)) = *None*
  assume *t*: *t*=*Stuck*
  assume *n*: *n* = *Suc m*
  from *no-bdy*
  have *exec-callC*:
    $\Gamma\vdash\langle$*Call* (*snd* (*cl s*)),*Normal* ((*upd* (*fst* (*cl s*)) $\circ$ *init*) *s*)$\rangle$ =*Suc m*$\Rightarrow$ *Stuck*
    by (*rule execn.CallUndefined*)
  from *execn.Seq* [*OF exec-upd* [*simplified n*]*exec-callC*]
  have *exec-closure*: $\Gamma\vdash$ $\langle$*callClosure upd* (*cl s*),*Normal* (*init s*)$\rangle$ =*n*$\Rightarrow$ *Stuck*
    by (*simp add*: *callClosure-def n*)
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-closure init-P'*] *t-notin-F t*
  have *False*
    by *auto*
  thus *?thesis* **..**
qed
qed

lemma *dynCallClosure*:
assumes *adapt*: $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F}\ P'$ (*callClosure upd* (*cl s*)) $Q',A' \wedge$
  init s $\in P' \wedge$
  $(\forall t \in Q'.$ *return s t* $\in R\ s\ t) \wedge$
  $(\forall t \in A'.$ *return s t* $\in A)\}$
assumes *res*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$

shows
$\Gamma,\Theta\vdash_{/F}$ P (*dynCallClosure init upd cl return c*) Q,A
  apply (*rule hoare-complete'*)
  apply (*rule allI*)
  apply (*rule dynCallClosure-sound* [**where** R=R])
  using *adapt*
  apply (*blast intro*: *hoare-cnvalid*)
  using *res*
  apply (*blast intro*: *hoare-cnvalid*)
  done

lemma *in-subsetD*: $[\![P \subseteq P'; x \in P]\!] \Longrightarrow x \in P'$
  by *blast*

lemma *dynCallClosureFix*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ cl'{=}cl\ s\ \wedge$
  $init\ s \in P'\ Z \ \wedge$
  $(\forall t \in Q'\ Z.\ return\ s\ t \in R\ s\ t) \ \wedge$
  $(\forall t \in A'\ Z.\ return\ s\ t \in A)\}$
assumes *res*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}$ $(R\ s\ t)$ $(c\ s\ t)$ Q,A
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $(P'\ Z)$ (*callClosure upd cl'*) $(Q'\ Z)$,$(A'\ Z)$
shows
$\Gamma,\Theta\vdash_{/F}$ P (*dynCallClosure init upd cl return c*) Q,A
  apply (*rule dynCallClosure* [*OF - res*])
  using *adapt spec*
  apply *clarsimp*
  apply (*drule* (*1*) *in-subsetD*)
  apply *clarsimp*
  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-extract-pre*:
  $[\![\forall s \in P.\ \Gamma,\Theta\vdash_{/F} (\{s\})\ c\ Q,A]\!]$
  $\Longrightarrow$
  $\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
  apply (*rule hoarep.Conseq*)
  apply *clarify*
  apply (*rule-tac* x=$\{s\}$ **in** *exI*)
  apply (*rule-tac* x=Q **in** *exI*)
  apply (*rule-tac* x=A **in** *exI*)
  by *simp*

lemma *app-closure-sound*:
  assumes *adapt*: $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P'\ (callClosure\ upd\ (e',p))\ Q',A' \wedge$
$upd\ x\ s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
  assumes *ap*: $upd\ e = upd\ e' \circ upd\ x$
  shows $\Gamma,\Theta \models n{:}_{/F}\ P\ (callClosure\ upd\ (e,p))\ Q,A$
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-e*: $\Gamma \vdash \langle callClosure\ upd\ (e,\ p), Normal\ s\rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *t*: $t \notin Fault$ ‘ F
  from *P adapt* **obtain** $P'\ Q'\ A'$
    where
    valid: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P'\ (callClosure\ upd\ (e',p))\ Q',A'$ **and**
    init-P': $upd\ x\ s \in P'$ **and**
    Q: $Q' \subseteq Q$ **and**
    A: $A' \subseteq A$
    by *auto*
  from *exec-e* [*simplified callClosure-def*] **obtain** s'
    where
    exec-e: $\Gamma \vdash \langle Basic\ (upd\ (fst\ (e,\ p))), Normal\ s\rangle =n\Rightarrow s'$**and**
    exec-p: $\Gamma \vdash \langle Call\ (snd\ (e,\ p)), s'\rangle =n\Rightarrow t$
    by *cases*
  from *exec-e* [*simplified*]
  have *s'*: $s' = Normal\ (upd\ e\ s)$
    by *cases simp*
  from *ap* **obtain** s'' **where**
   s'': $upd\ x\ s = s''$ **and** *upd-e'*: $upd\ e'\ s'' = upd\ e\ s$
    by *auto*
  from *ap s' execn.Basic* [**where** $f = (upd\ (fst\ (e',\ p)))$ **and** $s = upd\ x\ s$ **and** $\Gamma = \Gamma$]
  have *exec-e'*: $\Gamma \vdash \langle Basic\ (upd\ (fst\ (e',\ p))), Normal\ (upd\ x\ s)\rangle =n\Rightarrow s'$
    by *simp*
  with *exec-p*
  have $\Gamma \vdash \langle callClosure\ upd\ (e',\ p), Normal\ (upd\ x\ s)\rangle =n\Rightarrow t$
    by (*auto simp add*: *callClosure-def intro*: *execn.Seq*)
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt this init-P'*] *t Q A*
  show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A
    by *auto*
qed

lemma *app-closure*:
  assumes *adapt*: $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta \vdash_{/F}\ P'\ (callClosure\ upd\ (e',p))\ Q',A' \wedge$
$upd\ x\ s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
  assumes *ap*: $upd\ e = upd\ e' \circ upd\ x$
  shows $\Gamma,\Theta \vdash_{/F}\ P\ (callClosure\ upd\ (e,p))\ Q,A$
  apply (*rule hoare-complete'*)

apply (*rule allI*)
apply (*rule app-closure-sound* [**where** x=x **and** e'=e', *OF* - *ap*])
using *adapt*
apply (*blast intro*: *hoare-cnvalid*)
done

lemma *app-closure-spec*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ upd\ x\ s \in P'\ Z \wedge Q'\ Z \subseteq Q \wedge A'\ Z \subseteq A\}$
assumes *ap*: $upd\ e = upd\ e' \circ upd\ x$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (callClosure\ upd\ (e',p))\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F} P\ (callClosure\ upd\ (e,p))\ Q,A$
apply (*rule app-closure* [*OF* - *ap*])
apply *clarsimp*
using *adapt spec*
apply −
apply (*drule* (*1*) *in-subsetD*)
apply *clarsimp*
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

Implementation of closures as association lists.

definition $gen\text{-}upd\ var\ es\ s = foldl\ (\lambda s\ (x,i).\ the\ (var\ x)\ i\ s)\ s\ es$
definition $ap\ es\ c \equiv (es@fst\ c,snd\ c)$

lemma *gen-upd-app*: $\bigwedge es'.\ gen\text{-}upd\ var\ (es@es') = gen\text{-}upd\ var\ es' \circ gen\text{-}upd\ var\ es$
apply (*induct es*)
apply (*rule ext*)
apply (*simp add*: *gen-upd-def*)
apply (*rule ext*)
apply (*simp add*: *gen-upd-def*)
done

lemma *gen-upd-ap*:
$gen\text{-}upd\ var\ (fst\ (ap\ es\ (es',p))) = gen\text{-}upd\ var\ es' \circ gen\text{-}upd\ var\ es$
by (*simp add*: *gen-upd-app ap-def*)

lemma *ap-closure*:
assumes *adapt*: $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta\vdash_{/F} P'\ (callClosure\ (gen\text{-}upd\ var)\ c)\ Q',A' \wedge$
$gen\text{-}upd\ var\ es\ s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
shows $\Gamma,\Theta\vdash_{/F} P\ (callClosure\ (gen\text{-}upd\ var)\ (ap\ es\ c))\ Q,A$
proof −
obtain $es'\ p$ **where** c: c=(es',p)
by (*cases c*)
have $gen\text{-}upd\ var\ (fst\ (ap\ es\ (es',p))) = gen\text{-}upd\ var\ es' \circ gen\text{-}upd\ var\ es$

by (*simp add*: *gen-upd-ap*)
from *app-closure* [*OF adapt* [*simplified c*] *this*]
show *?thesis*
by (*simp add*: *c ap-def*)
qed

lemma *ap-closure-spec*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ gen\text{-}upd\ var\ es\ s \in P'\ Z \wedge Q'\ Z \subseteq Q \wedge A'\ Z \subseteq A\}$
assumes *spec*: $\forall Z.\ \Gamma,\Theta \vdash_{/F} (P'\ Z)\ (callClosure\ (gen\text{-}upd\ var)\ c)\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta \vdash_{/F} P\ (callClosure\ (gen\text{-}upd\ var)\ (ap\ es\ c))\ Q,A$
proof −
obtain *es' p* **where** *c*: *c*=(*es'*,*p*)
by (*cases c*)
have *gen-upd var* (*fst* (*ap es* (*es'*,*p*))) = *gen-upd var es'* ∘ *gen-upd var es*
by (*simp add*: *gen-upd-ap*)
from *app-closure-spec* [*OF adapt* [*simplified c*] *this spec* [*simplified c*]]
show *?thesis*
by (*simp add*: *c ap-def*)
qed

end

theory *ClosureEx*
imports ../*Vcg* ../*Simpl-Heap Closure*
begin

record *globals* =
cnt-' :: *ref* ⇒ *nat*
alloc-' :: *ref list*
free-' :: *nat*
record *'g vars* = *'g state* +
p-':: *ref*
r-':: *nat*
n-':: *nat*
m-':: *nat*
c-':: (*string* × *ref*) *list* × *string*
d-':: (*string* × *ref*) *list* × *string*
e-':: (*string* × *nat*) *list* × *string*

definition var_n = [''*n*''↦ (λ*x*. *n-'-update* (λ-. *x*)),
''*m*''↦ (λ*x*. *m-'-update* (λ-. *x*))]
definition upd_n = *gen-upd* var_n

lemma upd_n-ap: upd_n $(fst$ $(ap$ es $(es'$,$p)))$ = upd_n es' $\circ$ upd_n es
by (*simp add*: upd_n-*def gen-upd-ap*)

lemma
$\Gamma\vdash\{|\acute{}n=n_0 \wedge (\forall i\ j.\ \Gamma\vdash \{|\acute{}n=i \wedge \acute{}m=j|\}$ *callClosure* upd_n $\acute{}e$ $\{|\acute{}r=i + j|\})|\}$
$\acute{}e :== (ap\ [(''n'',\acute{}n)]\ \acute{}e)$
$\{|\forall j.\ \Gamma\vdash \{|\acute{}m=j|\}$ *callClosure* upd_n $\acute{}e$ $\{|\acute{}r=n_0 + j|\}|\}$
apply *vcg-step*
apply *clarify*
apply (*rule ap-closure* [**where** $var=var_n$, *folded* upd_n-*def*])
apply *clarsimp*
apply (*rename-tac s s'*)
apply (*erule-tac x=n-' s* **in** *allE*)
apply (*erule-tac x=m-' s'* **in** *allE*)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply (*assumption*)
apply (*simp add*: upd_n-*def gen-upd-def* var_n-*def*)
done

definition $var = [''p''\mapsto (\lambda x.\ p\text{-}'\text{-}update\ (\lambda\text{-}.\ x))]$
definition *upd* = *gen-upd var*

procedures $Inc(p|r)$ =
$\acute{}p\rightarrow\acute{}cnt :== \acute{}p\rightarrow\acute{}cnt + 1;;$
$\acute{}r :== \acute{}p\rightarrow\acute{}cnt$

lemma (**in** *Inc-impl*)
$\forall i\ p.\ \Gamma\vdash \{|\acute{}p\rightarrow\acute{}cnt = i|\}\ \acute{}r :== PROC\ Inc(\acute{}p)\ \{|\acute{}r=i+1 \wedge \acute{}p\rightarrow\acute{}cnt = i+1|\}$
apply *vcg*
apply *simp*
done

procedures (**imports** *Inc-signature*) $NewCounter(|c)$ =
$\acute{}p :== NEW\ 1\ [\acute{}cnt :== 0];;$
$\acute{}c :== ([(''p'',\acute{}p)],Inc\text{-}'proc)$

locale *NewCounter-impl'* = *NewCounter-impl* + *Inc-impl*
lemma (**in** *NewCounter-impl'*)
shows
$\forall alloc.\ \Gamma\vdash \{|1 \leq \acute{}free|\}\ \acute{}c :== PROC\ NewCounter()$
$\{|\exists p.\ p\rightarrow\acute{}cnt = 0\ \wedge$
$(\forall i.\ \Gamma\vdash \{|p\rightarrow\acute{}cnt = i|\}$ *callClosure upd* $\acute{}c$ $\{|\acute{}r=i+1 \wedge p\rightarrow\acute{}cnt = i+1|\})|\}$
apply *vcg*

apply *simp*
apply (*rule-tac x=new* (*set alloc*) **in** *exI*)
apply *simp*
apply (*simp add*: *callClosure-def*)
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply (*simp add*: *upd-def var-def gen-upd-def*)
done

lemma (**in** *NewCounter-impl'*)

shows
$\forall$ *alloc*. $\Gamma\vdash$ $\{\!|$*1* $\leq$ ´*free*$|\!\}$ ´*c* :== *PROC NewCounter*()
$\{\!|\exists$ *p*. *p*$\rightarrow$´*cnt* = *0* $\wedge$
($\forall$ *i*. $\Gamma\vdash$ $\{\!|$*p*$\rightarrow$´*cnt* = *i*$|\!\}$ *callClosure upd* ´*c* $\{\!|$´*r*=*i*+*1* $\wedge$ *p*$\rightarrow$´*cnt* = *i*+*1*$|\!\}$)$|\!\}$
apply *vcg*
apply *simp*
apply (*rule-tac x=new* (*set alloc*) **in** *exI*)
apply *simp*
apply (*simp add*: *callClosure-def*)
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply (*simp add*: *upd-def var-def gen-upd-def*)
done

lemma (**in** *NewCounter-impl'*)
shows *NewCounter-spec*:
$\forall$ *alloc*. $\Gamma\vdash$ $\{\!|$*1* $\leq$ ´*free* $\wedge$ ´*alloc*=*alloc*$|\!\}$ ´*c* :== *PROC NewCounter*()
$\{\!|\exists$ *p*. *p* $\notin$ *set alloc* $\wedge$ *p* $\in$ *set* ´*alloc* $\wedge$ *p* $\neq$ *Null* $\wedge$ *p*$\rightarrow$´*cnt* = *0* $\wedge$
($\forall$ *i*. $\Gamma\vdash$ $\{\!|$*p*$\rightarrow$´*cnt* = *i*$|\!\}$ *callClosure upd* ´*c* $\{\!|$´*r*=*i*+*1* $\wedge$ *p*$\rightarrow$´*cnt* = *i*+*1*$|\!\}$)$|\!\}$
apply *vcg*
apply *clarsimp*
apply (*rule-tac x=new* (*set alloc*) **in** *exI*)
apply *simp*
apply (*simp add*: *callClosure-def*)
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply (*simp add*: *upd-def var-def gen-upd-def*)
done

lemma $\Gamma\vdash$$\{\!|\exists$ *p*. *p* $\neq$ *Null* $\wedge$ *p*$\rightarrow$´*cnt* = *i* $\wedge$

$(\forall\, i.\ \Gamma\vdash \{\!|p\rightarrow\acute{}cnt = i|\!\}$ *callClosure upd* $\acute{}c\ \{\!|\acute{}r{=}i{+}1 \wedge p\rightarrow\acute{}cnt = i{+}1|\!\})|\!\}$
dynCallClosure $(\lambda s.\ s)$ *upd* $c\text{-}'$ $(\lambda s\ t.\ s(\!|globals := globals\ t|\!))$
$(\lambda s\ t.\ Basic\ (\lambda u.\ u(\!|r\text{-}' := r\text{-}'\ t|\!)))$
$\{\!|\acute{}r{=}i{+}1|\!\}$
apply (*rule conseq-extract-pre*)
apply *clarify*
apply (*rule dynCallClosureFix*)
apply (*simp only*: *Ball-def*)
prefer *3*
apply (*assumption*)
prefer *2*
apply *vcg-step*
apply *vcg-step*
apply (*simp only*: *simp-thms*)
apply *clarsimp*
done

lemma (**in** *NewCounter-impl'*)
shows $\Gamma\vdash \{\!|1 \le \acute{}free|\!\}$
$\acute{}c$:== *CALL NewCounter* ();;
dynCallClosure $(\lambda s.\ s)$ *upd* $c\text{-}'$ $(\lambda s\ t.\ s(\!|globals := globals\ t|\!))$
$(\lambda s\ t.\ Basic\ (\lambda u.\ u(\!|r\text{-}' := r\text{-}'\ t|\!)))$
$\{\!|\acute{}r{=}1|\!\}$
apply *vcg-step*
apply (*rule dynCallClosure*)
prefer *2*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply *clarsimp*
apply (*erule-tac x=0* **in** *allE*)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply (*assumption*)
apply *simp*
done

lemma (**in** *NewCounter-impl'*)
shows $\Gamma\vdash \{\!|1 \le \acute{}free|\!\}$
$\acute{}c$:== *CALL NewCounter* ();;
dynCallClosure $(\lambda s.\ s)$ *upd* $c\text{-}'$ $(\lambda s\ t.\ s(\!|globals := globals\ t|\!))$
$(\lambda s\ t.\ Basic\ (\lambda u.\ u(\!|r\text{-}' := r\text{-}'\ t|\!)))$;;
dynCallClosure $(\lambda s.\ s)$ *upd* $c\text{-}'$ $(\lambda s\ t.\ s(\!|globals := globals\ t|\!))$
$(\lambda s\ t.\ Basic\ (\lambda u.\ u(\!|r\text{-}' := r\text{-}'\ t|\!)))$
$\{\!|\acute{}r{=}2|\!\}$
apply *vcg-step*
apply (*rule dynCallClosure*)

prefer *2*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply (*rule dynCallClosure*)
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*

apply *clarsimp*
apply (*subgoal-tac* $\Gamma\vdash$ $\{\!|p\rightarrow$´$cnt = 0|\!\}$ *callClosure upd* (*c-*′ *t*) $\{\!|$´$r = Suc\ 0 \wedge p\rightarrow$´$cnt = Suc\ 0|\!\}$)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply *assumption*
apply *clarsimp*
apply (*erule-tac x=1* **in** *allE*)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply *assumption*
apply *clarsimp*
apply (*erule allE*)
apply *assumption*
done

lemma (**in** *NewCounter-impl*′)
shows $\Gamma\vdash$ $\{\!|1 \le$ ´$free|\!\}$
´c :== *CALL NewCounter* ();;
´d :== ´c;;
dynCallClosure ($\lambda s.\ s$) *upd c-*′ ($\lambda s\ t.\ s(\!|globals := globals\ t|\!)$)
($\lambda s\ t.$ *Basic* ($\lambda u.\ u(\!|n\text{-}' := r\text{-}'\ t|\!)$));;
dynCallClosure ($\lambda s.\ s$) *upd d-*′ ($\lambda s\ t.\ s(\!|globals := globals\ t|\!)$)
($\lambda s\ t.$ *Basic* ($\lambda u.\ u(\!|m\text{-}' := r\text{-}'\ t|\!)$));;
´r :== ´n + ´m
$\{\!|$´$r=3|\!\}$

apply *vcg-step*
apply *vcg-step*
apply (*rule dynCallClosure*)
prefer *2*
apply *vcg-step*
apply *vcg-step*
apply *vcg-step*
apply (*rule dynCallClosure*)
apply *vcg-step*
apply *vcg-step*

apply *vcg-step*
apply *vcg-step*
apply *clarsimp*
apply (*subgoal-tac* $\Gamma\vdash$ $\{\!|p\rightarrow\acute{}cnt = 0|\!\}$ *callClosure upd* (*c-*′ *t*) $\{\!|\acute{}r = Suc\ 0 \wedge p\rightarrow\acute{}$ $cnt = Suc\ 0|\!\}$)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply *assumption*
apply *clarsimp*
apply (*erule-tac x=1* **in** *allE*)
apply (*rule exI*)
apply (*rule exI*)
apply (*rule conjI*)
apply *assumption*
apply *clarsimp*
apply (*erule allE*)
apply *assumption*
done

end

25 Experiments on State Composition

theory *Compose* **imports** ../*HoareTotalProps* **begin**

We develop some theory to support state-space modular development of programs. These experiments aim at the representation of state-spaces with records. If we use *statespaces* instead we get this kind of compositionality for free.

25.1 Changing the State-Space

definition $lift_f$:: $('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('S \Rightarrow 'S)$
  where $lift_f$ *prj inject f* = ($\lambda S.$ *inject S* (*f* (*prj S*)))

definition $lift_s$:: $('S \Rightarrow 's) \Rightarrow 's\ set \Rightarrow 'S\ set$
  where $lift_s$ *prj A* = $\{S.\ prj\ S \in A\}$

definition $lift_r$:: $('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s \times 's)\ set$
    $\Rightarrow ('S \times 'S)\ set$
where
$lift_r$ *prj inject R* = $\{(S,T).\ (prj\ S, prj\ T) \in R \wedge T = inject\ S\ (prj\ T)\}$

primrec $lift_c$:: $('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s,'p,'f)\ com \Rightarrow ('S,'p,'f)\ com$
where
$lift_c$ *prj inject Skip* = *Skip* |

$lift_c$ *prj inject* (*Basic f*) = *Basic* ($lift_f$ *prj inject f*) |
$lift_c$ *prj inject* (*Spec r*) = *Spec* ($lift_r$ *prj inject r*) |
$lift_c$ *prj inject* (*Seq* c_1 c_2) =
 (*Seq* ($lift_c$ *prj inject* c_1) ($lift_c$ *prj inject* c_2)) |
$lift_c$ *prj inject* (*Cond b* c_1 c_2) =
 Cond ($lift_s$ *prj b*) ($lift_c$ *prj inject* c_1) ($lift_c$ *prj inject* c_2) |
$lift_c$ *prj inject* (*While b c*) =
 While ($lift_s$ *prj b*) ($lift_c$ *prj inject c*) |
$lift_c$ *prj inject* (*Call p*) = *Call p* |
$lift_c$ *prj inject* (*DynCom c*) = *DynCom* ($\lambda s.$ $lift_c$ *prj inject* (*c* (*prj s*))) |
$lift_c$ *prj inject* (*Guard f g c*) = *Guard f* ($lift_s$ *prj g*) ($lift_c$ *prj inject c*) |
$lift_c$ *prj inject Throw* = *Throw* |
$lift_c$ *prj inject* (*Catch* c_1 c_2) =
 Catch ($lift_c$ *prj inject* c_1) ($lift_c$ *prj inject* c_2)

lemma $lift_c$-*Skip*: ($lift_c$ *prj inject c* = *Skip*) = (*c* = *Skip*)
 by (*cases c*) *auto*

lemma $lift_c$-*Basic*:
 ($lift_c$ *prj inject c* = *Basic lf*) = ($\exists f.$ *c* = *Basic f* $\wedge$ *lf* = $lift_f$ *prj inject f*)
 by (*cases c*) *auto*

lemma $lift_c$-*Spec*:
 ($lift_c$ *prj inject c* = *Spec lr*) = ($\exists r.$ *c* = *Spec r* $\wedge$ *lr* = $lift_r$ *prj inject r*)
 by (*cases c*) *auto*

lemma $lift_c$-*Seq*:
 ($lift_c$ *prj inject c* = *Seq* lc_1 lc_2) =
  ($\exists$ c_1 c_2. *c* = *Seq* c_1 c_2 $\wedge$
   lc_1 = $lift_c$ *prj inject* c_1 $\wedge$ lc_2 = $lift_c$ *prj inject* c_2)
  by (*cases c*) *auto*

lemma $lift_c$-*Cond*:
 ($lift_c$ *prj inject c* = *Cond lb* lc_1 lc_2) =
  ($\exists$ *b* c_1 c_2. *c* = *Cond b* c_1 c_2 $\wedge$ *lb* = $lift_s$ *prj b* $\wedge$
   lc_1 = $lift_c$ *prj inject* c_1 $\wedge$ lc_2 = $lift_c$ *prj inject* c_2)
 by (*cases c*) *auto*

lemma $lift_c$-*While*:
 ($lift_c$ *prj inject c* = *While lb* lc') =
  ($\exists$ *b* c'. *c* = *While b* c' $\wedge$ *lb* = $lift_s$ *prj b* $\wedge$
   lc' = $lift_c$ *prj inject* c')
 by (*cases c*) *auto*

lemma $lift_c$-*Call*:
 ($lift_c$ *prj inject c* = *Call p*) = (*c* = *Call p*)
 by (*cases c*) *auto*

lemma $lift_c$*-DynCom*:
  $(lift_c\ prj\ inject\ c = DynCom\ lc) =$
    $(\exists\, C.\ c{=}DynCom\ C \wedge lc = (\lambda s.\ lift_c\ prj\ inject\ (C\ (prj\ s))))$
  by (*cases c*) *auto*

lemma $lift_c$*-Guard*:
  $(lift_c\ prj\ inject\ c = Guard\ f\ lg\ lc') =$
    $(\exists\, g\ c'.\ c = Guard\ f\ g\ c' \wedge lg = lift_s\ prj\ g \wedge$
        $lc' = lift_c\ prj\ inject\ c')$
   by (*cases c*) *auto*

lemma $lift_c$*-Throw*:
  $(lift_c\ prj\ inject\ c = Throw) = (c = Throw)$
  by (*cases c*) *auto*

lemma $lift_c$*-Catch*:
  $(lift_c\ prj\ inject\ c = Catch\ lc_1\ lc_2) =$
    $(\exists\ c_1\ c_2.\ c = Catch\ c_1\ c_2 \wedge$
        $lc_1 = lift_c\ prj\ inject\ c_1 \wedge lc_2 = lift_c\ prj\ inject\ c_2\ )$
   by (*cases c*) *auto*

definition *xstate-map*:: $('S \Rightarrow 's) \Rightarrow ('S,'f)\ xstate \Rightarrow ('s,'f)\ xstate$
where
xstate-map g x = (*case x of*
      Normal s ⇒ *Normal* (*g s*)
    | *Abrupt s* ⇒ *Abrupt* (*g s*)
    | *Fault f* ⇒ *Fault f*
    | *Stuck* ⇒ *Stuck*)

lemma *xstate-map-simps* [*simp*]:
xstate-map g (*Normal s*) = *Normal* (*g s*)
xstate-map g (*Abrupt s*) = *Abrupt* (*g s*)
xstate-map g (*Fault f*) = (*Fault f*)
xstate-map g Stuck = *Stuck*
  by (*auto simp add*: *xstate-map-def*)

lemma *xstate-map-Normal-conv*:
  xstate-map g S = *Normal s* = $(\exists\, s'.\ S{=}Normal\ s' \wedge s = g\ s')$
  by (*cases S*) *auto*

lemma *xstate-map-Abrupt-conv*:
  xstate-map g S = *Abrupt s* = $(\exists\, s'.\ S{=}Abrupt\ s' \wedge s = g\ s')$
  by (*cases S*) *auto*

lemma *xstate-map-Fault-conv*:
  xstate-map g S = *Fault f* = (*S*=*Fault f*)

by (*cases S*) *auto*

lemma *xstate-map-Stuck-conv*:
xstate-map g S = Stuck = (S=Stuck)
by (*cases S*) *auto*

lemmas *xstate-map-convs = xstate-map-Normal-conv xstate-map-Abrupt-conv xstate-map-Fault-conv xstate-map-Stuck-conv*

definition *state*:: $('s,'f)$ *xstate* $\Rightarrow$ $'s$
where
state x = (*case x of*
Normal s $\Rightarrow$ *s*
| *Abrupt s* $\Rightarrow$ *s*
| *Fault g* $\Rightarrow$ *undefined*
| *Stuck* $\Rightarrow$ *undefined*)

lemma *state-simps* [*simp*]:
state (*Normal s*) = *s*
state (*Abrupt s*) = *s*
by (*auto simp add*: *state-def*)

locale *lift-state-space* =
fixes *project*::$'S \Rightarrow 's$
fixes *inject*::$'S \Rightarrow 's \Rightarrow 'S$
fixes $project_x$::$('S,'f)$ *xstate* $\Rightarrow$ $('s,'f)$ *xstate*
fixes $lift_e$::$('s,'p,'f)$ *body* $\Rightarrow$ $('S,'p,'f)$ *body*
fixes $lift_c$:: $('s,'p,'f)$ *com* $\Rightarrow$ $('S,'p,'f)$ *com*
fixes $lift_f$:: $('s \Rightarrow 's) \Rightarrow ('S \Rightarrow 'S)$
fixes $lift_s$:: $'s$ *set* $\Rightarrow$ $'S$ *set*
fixes $lift_r$:: $('s \times 's)$ *set* $\Rightarrow$ $('S \times 'S)$ *set*
assumes *proj-inj-commute*: $\bigwedge S\ s.$ *project* (*inject S s*) = *s*
defines $lift_c \equiv Compose.lift_c$ *project inject*
defines $project_x \equiv$ *xstate-map project*
defines $lift_e \equiv (\lambda\Gamma\ p.$ *map-option* $lift_c$ $(\Gamma\ p))$
defines $lift_f \equiv Compose.lift_f$ *project inject*
defines $lift_s \equiv Compose.lift_s$ *project*
defines $lift_r \equiv Compose.lift_r$ *project inject*

lemma (**in** *lift-state-space*) $lift_f$*-simp*:
$lift_f\ f \equiv \lambda S.$ *inject S* (*f* (*project S*))
by (*simp add*: $lift_f$*-def* $Compose.lift_f$*-def*)

lemma (**in** *lift-state-space*) $lift_s$*-simp*:
$lift_s\ A \equiv \{S.$ *project* $S \in A\}$
by (*simp add*: $lift_s$*-def* $Compose.lift_s$*-def*)

lemma (**in** *lift-state-space*) $lift_r$*-simp*:
$lift_r\ R \equiv \{(S,T).\ (project\ S, project\ T) \in R \wedge T{=}inject\ S\ (project\ T)\}$
by (*simp add*: $lift_r$*-def Compose.*$lift_r$*-def*)

lemma (**in** *lift-state-space*) $lift_c$*-Skip-simp* [*simp*]:
$lift_c\ Skip = Skip$
by (*simp add*: $lift_c$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Basic-simp* [*simp*]:
$lift_c\ (Basic\ f) = Basic\ (lift_f\ f)$
by (*simp add*: $lift_c$*-def* $lift_f$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Spec-simp* [*simp*]:
$lift_c\ (Spec\ r) = Spec\ (lift_r\ r)$
by (*simp add*: $lift_c$*-def* $lift_r$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Seq-simp* [*simp*]:
$lift_c\ (Seq\ c_1\ c_2) =$
$(Seq\ (lift_c\ c_1)\ (lift_c\ c_2))$
by (*simp add*: $lift_c$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Cond-simp* [*simp*]:
$lift_c\ (Cond\ b\ c_1\ c_2) =$
$Cond\ (lift_s\ b)\ (lift_c\ c_1)\ (lift_c\ c_2)$
by (*simp add*: $lift_c$*-def* $lift_s$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-While-simp* [*simp*]:
$lift_c\ (While\ b\ c) =$
$While\ (lift_s\ b)\ (lift_c\ c)$
by (*simp add*: $lift_c$*-def* $lift_s$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Call-simp* [*simp*]:
$lift_c\ (Call\ p) = Call\ p$
by (*simp add*: $lift_c$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-DynCom-simp* [*simp*]:
$lift_c\ (DynCom\ c) = DynCom\ (\lambda s.\ lift_c\ (c\ (project\ s)))$
by (*simp add*: $lift_c$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Guard-simp* [*simp*]:
$lift_c\ (Guard\ f\ g\ c) = Guard\ f\ (lift_s\ g)\ (lift_c\ c)$
by (*simp add*: $lift_c$*-def* $lift_s$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Throw-simp* [*simp*]:
$lift_c\ Throw = Throw$
by (*simp add*: $lift_c$*-def*)
lemma (**in** *lift-state-space*) $lift_c$*-Catch-simp* [*simp*]:
$lift_c\ (Catch\ c_1\ c_2) =$
$Catch\ (lift_c\ c_1)\ (lift_c\ c_2)$
by (*simp add*: $lift_c$*-def*)

lemma (**in** *lift-state-space*) $project_x$*-def'*:
$project_x\ s \equiv (case\ s\ of$
$Normal\ s \Rightarrow Normal\ (project\ s)$
$\mid Abrupt\ s \Rightarrow Abrupt\ (project\ s)$
$\mid Fault\ f \Rightarrow Fault\ f$
$\mid Stuck \Rightarrow Stuck)$

by (*simp add*: *xstate-map-def* $project_x$*-def*)

lemma (**in** *lift-state-space*) $lift_e$*-def* ′:
$lift_e$ Γ *p* ≡ (*case* Γ *p of Some bdy* ⇒ *Some* ($lift_c$ *bdy*) | *None* ⇒ *None*)
by (*simp add*: $lift_e$*-def map-option-case*)

The problem is that $lift_c$ *project inject* ∘ Γ is quite a strong premise. The problem is that Γ is a function here. A map would be better. We only have to lift those procedures in the domain of Γ: Γ *p* = *Some bdy* ⟶ Γ′ *p* = *Some* $lift_c$ *project inject bdy*. We then can com up with theorems that allow us to extend the domains of Γ and preserve validity.

lemma (**in** *lift-state-space*)
{(*S*,*T*). ∃ *t*. (*project S*,*t*) ∈ *r* ∧ *T*=*inject S t*}
⊆ {(*S*,*T*). (*project S*,*project T*) ∈ *r* ∧ *T*=*inject S* (*project T*)}
apply *clarsimp*
apply (*rename-tac S t*)
apply (*simp add*: *proj-inj-commute*)
done

lemma (**in** *lift-state-space*)
{(*S*,*T*). (*project S*,*project T*) ∈ *r* ∧ *T*=*inject S* (*project T*)}
⊆ {(*S*,*T*). ∃ *t*. (*project S*,*t*) ∈ *r* ∧ *T*=*inject S t*}
apply *clarsimp*
apply (*rename-tac S T*)
apply (*rule-tac x*=*project T* **in** *exI*)
apply *simp*
done

lemma (**in** *lift-state-space*) *lift-exec*:
assumes *exec-lc*: ($lift_e$ Γ)⊢⟨*lc*,*s*⟩ ⇒ *t*
shows ⋀*c*. ⟦ $lift_c$ *c* = *lc*⟧ ⟹
Γ⊢⟨*c*,$project_x$ *s*⟩ ⇒ $project_x$ *t*
using *exec-lc*
proof (*induct*)
case *Skip* **thus** *?case*
by (*auto simp add*: $project_x$*-def* $lift_c$*-Skip* $lift_c$*-def intro*: *exec.Skip*)
next
case *Guard* **thus** *?case*
by (*auto simp add*: $project_x$*-def* $lift_s$*-def Compose.*$lift_s$*-def* $lift_c$*-Guard* $lift_c$*-def intro*: *exec.Guard*)
next
case *GuardFault* **thus** *?case*
by (*auto simp add*: $project_x$*-def* $lift_s$*-def Compose.*$lift_s$*-def* $lift_c$*-Guard* $lift_c$*-def intro*: *exec.GuardFault*)
next
case *FaultProp* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def*)
next

case *Basic*
thus *?case*
by (*fastforce simp add*: *project*$_x$*-def lift*$_c$*-Basic lift*$_f$*-def Compose.lift*$_f$*-def*
lift$_c$*-def*
proj-inj-commute
intro: *exec.Basic*)
next
case *Spec*
thus *?case*
by (*fastforce simp add*: *project*$_x$*-def lift*$_c$*-Spec lift*$_f$*-def Compose.lift*$_f$*-def*
lift$_r$*-def Compose.lift*$_r$*-def lift*$_c$*-def*
proj-inj-commute
intro: *exec.Spec*)
next
case (*SpecStuck s r*)
thus *?case*
apply (*simp add*: *project*$_x$*-def*)
apply (*clarsimp simp add*: *lift*$_c$*-Spec lift*$_c$*-def*)
apply (*unfold lift*$_r$*-def Compose.lift*$_r$*-def*)
apply (*rule exec.SpecStuck*)
apply (*rule allI*)
apply (*erule-tac x=inject s t* **in** *allE*)
apply *clarsimp*
apply (*simp add*: *proj-inj-commute*)
done
next
case *Seq*
thus *?case*
by (*fastforce simp add*: *project*$_x$*-def lift*$_c$*-Seq lift*$_c$*-def intro*: *exec.intros*)
next
case *CondTrue*
thus *?case*
by (*auto simp add*: *project*$_x$*-def lift*$_s$*-def Compose.lift*$_s$*-def lift*$_c$*-Cond lift*$_c$*-def*
intro: *exec.CondTrue*)
next
case *CondFalse*
thus *?case*
by (*auto simp add*: *project*$_x$*-def lift*$_s$*-def Compose.lift*$_s$*-def lift*$_c$*-Cond lift*$_c$*-def*
intro: *exec.CondFalse*)
next
case *WhileTrue*
thus *?case*
by (*fastforce simp add*: *project*$_x$*-def lift*$_s$*-def Compose.lift*$_s$*-def*
lift$_c$*-While lift*$_c$*-def*
intro: *exec.WhileTrue*)
next
case *WhileFalse*
thus *?case*
by (*fastforce simp add*: *project*$_x$*-def lift*$_s$*-def Compose.lift*$_s$*-def*

lift$_c$-*While lift*$_c$-*def*
intro: *exec.WhileFalse*)

next
case *Call*
thus *?case*
by (*fastforce simp add*:
project$_x$-*def lift*$_c$-*Call lift*$_f$-*def Compose.lift*$_f$-*def lift*$_c$-*def*
lift$_e$-*def*
intro: *exec.Call*)

next
case *CallUndefined*
thus *?case*
by (*fastforce simp add*:
project$_x$-*def lift*$_c$-*Call lift*$_f$-*def Compose.lift*$_f$-*def lift*$_c$-*def*
lift$_e$-*def*
intro: *exec.CallUndefined*)

next
case *StuckProp* **thus** *?case*
by (*fastforce simp add*: *project*$_x$-*def*)

next
case *DynCom*
thus *?case*
by (*fastforce simp add*:
project$_x$-*def lift*$_c$-*DynCom lift*$_f$-*def Compose.lift*$_f$-*def lift*$_c$-*def*
intro: *exec.DynCom*)

next
case *Throw* **thus** *?case*
by (*fastforce simp add*: *project*$_x$-*def lift*$_c$-*Throw lift*$_c$-*def intro*: *exec.Throw*)

next
case *AbruptProp* **thus** *?case*
by (*fastforce simp add*: *project*$_x$-*def*)

next
case *CatchMatch*
thus *?case*
by (*fastforce simp add*: *project*$_x$-*def lift*$_c$-*Catch lift*$_c$-*def intro*: *exec.CatchMatch*)

next
case (*CatchMiss* c_1 s t c_2 c)
thus *?case*
by (*cases* t)
(*fastforce simp add*: *project*$_x$-*def lift*$_c$-*Catch lift*$_c$-*def intro*: *exec.CatchMiss*)+

qed

lemma (**in** *lift-state-space*) *lift-exec*′:
assumes *exec-lc*: (*lift*$_e$ Γ)$\vdash\langle$*lift*$_c$ c,$s\rangle \Rightarrow t$
shows $\Gamma\vdash\langle c$,*project*$_x$ $s\rangle \Rightarrow$ *project*$_x$ t
using *lift-exec* [*OF exec-lc*]
by *simp*

lemma (**in** *lift-state-space*) *lift-valid*:
assumes *valid*: $\Gamma\models_{/F}$ P c Q,A
shows
($lift_e$ Γ)$\models_{/F}$ ($lift_s$ P) ($lift_c$ c) ($lift_s$ Q),($lift_s$ A)
proof (*rule validI*)
fix s t
assume *lexec*:
($lift_e$ Γ)$\vdash\langle lift_c$ c,*Normal* $s\rangle \Rightarrow t$
assume *lP*: $s \in lift_s$ P
assume *noFault*: $t \notin$ *Fault* ‘ F
show $t \in$ *Normal* ‘ $lift_s$ Q $\cup$ *Abrupt* ‘ $lift_s$ A
proof −
from *lexec*
have $\Gamma\vdash$ $\langle c$,$project_x$ (*Normal* s)$\rangle \Rightarrow$ ($project_x$ t)
by (*rule lift-exec*) (*simp-all*)
moreover
from *lP* **have** *project* $s \in P$
by (*simp add*: $lift_s$-*def Compose*.$lift_s$-*def* $project_x$-*def*)
ultimately
have $project_x$ $t \in$ *Normal* ‘ Q $\cup$ *Abrupt* ‘ A
using *valid noFault*
apply (*clarsimp simp add*: *valid-def* $project_x$-*def*)
apply (*cases* t)
apply *auto*
done
thus *?thesis*
apply (*simp add*: $lift_s$-*def Compose*.$lift_s$-*def*)
apply (*cases* t)
apply (*auto simp add*: $project_x$-*def*)
done
qed
qed

lemma (**in** *lift-state-space*) *lift-hoarep*:
assumes *deriv*: Γ,{}$\vdash_{/F}$ P c Q,A
shows
($lift_e$ Γ),{}$\vdash_{/F}$ ($lift_s$ P) ($lift_c$ c) ($lift_s$ Q),($lift_s$ A)
apply (*rule hoare-complete*)
apply (*insert hoare-sound* [*OF deriv*])
apply (*rule lift-valid*)
apply (*simp add*: *cvalid-def*)
done

lemma (**in** *lift-state-space*) *lift-hoarep'*:
$\forall Z$. Γ,{}$\vdash_{/F}$ (P Z) c (Q Z),(A Z) $\Longrightarrow$
$\forall Z$. ($lift_e$ Γ),{}$\vdash_{/F}$ ($lift_s$ (P Z)) ($lift_c$ c)
($lift_s$ (Q Z)),($lift_s$ (A Z))

apply (*iprover intro*: *lift-hoarep*)
done

lemma (**in** *lift-state-space*) *lift-termination*:
assumes *termi*: $\Gamma\vdash c\downarrow s$
shows $\bigwedge S.$ *project*$_x$ $S = s \Longrightarrow$
  lift$_e$ $\Gamma \vdash$(*lift*$_c$ *c*)$\downarrow S$
  using *termi*
proof (*induct*)
  case *Skip* **thus** *?case*
    by (*clarsimp simp add*: *terminates.Skip project*$_x$*-def xstate-map-convs*)
next
  case *Basic* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs intro*: *terminates.intros*)
next
  case *Guard* **thus** *?case*
    by (*auto simp add*: *project*$_x$*-def xstate-map-convs intro*: *terminates.intros*)
next
  case *GuardFault* **thus** *?case*
    by (*auto simp add*: *project*$_x$*-def xstate-map-convs lift*$_s$*-def Compose.lift*$_s$*-def*
        intro: *terminates.intros*)
next
  case *Fault* **thus** *?case* **by** (*clarsimp simp add*: *project*$_x$*-def xstate-map-convs*)
next
  case (*Seq c1 s c2*)
  have *project*$_x$ $S =$ *Normal s* **by** *fact*
  then obtain *s'* **where** *S*: *S=Normal s'* **and** *s*: *s = project s'*
    by (*auto simp add*: *project*$_x$*-def xstate-map-convs*)
  from *Seq* **have** *lift*$_e$ $\Gamma\vdash$*lift*$_c$ *c1* $\downarrow$ *S*
    by *simp*
  moreover
  {
    fix *w*
    assume *exec-lc1*: *lift*$_e$ $\Gamma\vdash\langle$*lift*$_c$ *c1*,*Normal s'*$\rangle \Rightarrow w$
    have *lift*$_e$ $\Gamma\vdash$*lift*$_c$ *c2* $\downarrow$ *w*
    proof (*cases w*)
      case (*Normal w'*)
      with *lift-exec* [**where** *c=c1*, *OF exec-lc1*] *s*
      have $\Gamma\vdash\langle$*c1*,*Normal s*$\rangle \Rightarrow$ *Normal* (*project w'*)
        by (*simp add*: *project*$_x$*-def*)
      from *Seq.hyps* (*3*) [*rule-format*, *OF this*] *Normal*
      show *lift*$_e$ $\Gamma\vdash$*lift*$_c$ *c2* $\downarrow$ *w*
        by (*auto simp add*: *project*$_x$*-def xstate-map-convs*)
    qed (*auto*)

}
ultimately show *?case*
using *S s*
by (*auto intro*: *terminates.intros*)
next
case *CondTrue* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def* $lift_s$*-def* *Compose.*$lift_s$*-def xstate-map-convs*
intro: *terminates.intros*)
next
case *CondFalse* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def* $lift_s$*-def* *Compose.*$lift_s$*-def xstate-map-convs*
intro: *terminates.intros*)
next
case (*WhileTrue s b c*)
have $project_x$ *S* = *Normal s* **by** *fact*
then obtain *s'* **where** *S*: *S*=*Normal s'* **and** *s*: *s* = *project s'*
by (*auto simp add*: $project_x$*-def xstate-map-convs*)
from *WhileTrue* **have** $lift_e$ Γ⊢$lift_c$ *c* ↓ *S*
by *simp*
moreover
{
fix *w*
assume *exec-lc*: $lift_e$ Γ⊢⟨$lift_c$ *c*,*Normal s'*⟩ ⇒ *w*
have $lift_e$ Γ⊢$lift_c$ (*While b c*) ↓ *w*
proof (*cases w*)
case (*Normal w'*)
with *lift-exec* [**where** *c*=*c*, *OF exec-lc*] *s*
have Γ⊢⟨*c*,*Normal s*⟩ ⇒ *Normal* (*project w'*)
by (*simp add*: $project_x$*-def*)
from *WhileTrue.hyps* (*4*) [*rule-format*, *OF this*] *Normal*
show $lift_e$ Γ⊢$lift_c$ (*While b c*) ↓ *w*
by (*auto simp add*: $project_x$*-def xstate-map-convs*)
qed (*auto*)
}
ultimately show *?case*
using *S s*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def* $lift_s$*-def* *Compose.*$lift_s$*-def xstate-map-convs*
intro: *terminates.intros*)
next
case *Call* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def xstate-map-convs* $lift_e$*-def*
intro: *terminates.intros*)
next
case *CallUndefined* **thus** *?case*
by (*fastforce simp add*: $project_x$*-def xstate-map-convs* $lift_e$*-def*
intro: *terminates.intros*)

next
  case *Stuck* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs*)
next
  case *DynCom* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs*
      intro: *terminates.intros*)
next
  case *Throw* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs*
      intro: *terminates.intros*)
next
  case *Abrupt* **thus** *?case*
    by (*fastforce simp add*: *project*$_x$*-def xstate-map-convs*
      intro: *terminates.intros*)
next
  case (*Catch c1 s c2*)
  have *project*$_x$ *S* = *Normal s* **by** *fact*
  then obtain *s'* **where** *S*: *S*=*Normal s'* **and** *s*: *s* = *project s'*
    by (*auto simp add*: *project*$_x$*-def xstate-map-convs*)
  from *Catch* **have** *lift*$_e$ Γ⊢*lift*$_c$ *c1* ↓ *S*
    by *simp*
  moreover
  {
    fix *w*
    assume *exec-lc1*: *lift*$_e$ Γ⊢⟨*lift*$_c$ *c1*,*Normal s'*⟩ ⇒ *Abrupt w*
    have *lift*$_e$ Γ⊢*lift*$_c$ *c2* ↓ *Normal w*
    proof −
      from *lift-exec* [**where** *c*=*c1*, *OF exec-lc1*] *s*
      have Γ⊢⟨*c1*,*Normal s*⟩ ⇒ *Abrupt* (*project w*)
        by (*simp add*: *project*$_x$*-def*)
      from *Catch.hyps* (*3*) [*rule-format*, *OF this*]
      show *lift*$_e$ Γ⊢*lift*$_c$ *c2* ↓ *Normal w*
        by (*auto simp add*: *project*$_x$*-def xstate-map-convs*)
    qed
  }
  ultimately show *?case*
    using *S s*
    by (*auto intro*: *terminates.intros*)
qed

lemma (**in** *lift-state-space*) *lift-termination'*:
assumes *termi*: Γ⊢*c*↓*project*$_x$ *S*
shows *lift*$_e$ Γ ⊢(*lift*$_c$ *c*)↓*S*
  using *lift-termination* [*OF termi*]
  by *iprover*

lemma (**in** *lift-state-space*) *lift-validt*:

assumes *valid*: $\Gamma\models_{t/F}$ *P c Q*,*A*
shows (*lift*$_e$ Γ)$\models_{t/F}$ (*lift*$_s$ *P*) (*lift*$_c$ *c*) (*lift*$_s$ *Q*),(*lift*$_s$ *A*)
proof −
from *valid*
have (*lift*$_e$ Γ)$\models_{/F}$ (*lift*$_s$ *P*) (*lift*$_c$ *c*) (*lift*$_s$ *Q*),(*lift*$_s$ *A*)
by (*auto intro*: *lift-valid simp add*: *validt-def*)
moreover
{
fix *S*
assume $S \in$ *lift*$_s$ *P*
hence *project* $S \in P$
by (*simp add*: *lift*$_s$*-def Compose.lift*$_s$*-def*)
with *valid* **have** $\Gamma\vdash c \downarrow$ *project*$_x$ (*Normal S*)
by (*simp add*: *validt-def project*$_x$*-def*)
hence *lift*$_e$ $\Gamma\vdash$*lift*$_c$ $c \downarrow$ *Normal S*
by (*rule lift-termination*$'$)
}
ultimately show *?thesis*
by (*simp add*: *validt-def*)
qed

lemma (**in** *lift-state-space*) *lift-hoaret*:
assumes *deriv*: Γ,{}$\vdash_{t/F}$ *P c Q*,*A*
shows
(*lift*$_e$ Γ),{}$\vdash_{t/F}$ (*lift*$_s$ *P*) (*lift*$_c$ *c*) (*lift*$_s$ *Q*),(*lift*$_s$ *A*)
apply (*rule hoaret-complete*)
apply (*insert hoaret-sound* [*OF deriv*])
apply (*rule lift-validt*)
apply (*simp add*: *cvalidt-def*)
done

locale *lift-state-space-ext* = *lift-state-space* +
assumes *inj-proj-commute*: $\bigwedge S.$ *inject S* (*project S*) = *S*
assumes *inject-last*: $\bigwedge S\ s\ t.$ *inject* (*inject S s*) *t* = *inject S t*

lemma (**in** *lift-state-space-ext*) *lift-exec-inject-same*:
assumes *exec-lc*: (*lift*$_e$ Γ)$\vdash\langle lc,s\rangle \Rightarrow t$
shows $\bigwedge c.$ $[\![$*lift*$_c$ *c* = *lc*; $t \notin$ (*Fault* ‘ *UNIV*) $\cup$ {*Stuck*}$]\!] \Longrightarrow$
state t = *inject* (*state s*) (*project* (*state t*))
using *exec-lc*
proof (*induct*)
case *Skip* **thus** *?case*
by (*clarsimp simp add*: *inj-proj-commute*)
next
case *Guard* **thus** *?case*

    by (*clarsimp simp add*: *lift*$_c$-*Guard lift*$_c$-*def*)
next
  case *GuardFault* **thus** *?case*
    by *simp*
next
  case *FaultProp* **thus** *?case* **by** *simp*
next
  case *Basic* **thus** *?case*
    by (*clarsimp simp add*: *lift*$_f$-*def Compose.lift*$_f$-*def*
        proj-inj-commute lift$_c$-*Basic lift*$_c$-*def*)
next
  case (*Spec r*) **thus** *?case*
    by (*clarsimp simp add*: *Compose.lift*$_r$-*def lift*$_c$-*Spec lift*$_c$-*def*)
next
  case *SpecStuck*
  thus *?case* **by** *simp*
next
  case (*Seq lc1 s s' lc2 t c*)
  have *t*: $t \notin$ *Fault* ` *UNIV* $\cup$ {*Stuck*} **by** *fact*
  have *lift*$_c$ *c* = *Seq lc1 lc2* **by** *fact*
  then obtain *c1 c2* **where**
    c: *c* = *Seq c1 c2* **and**
    lc1: *lc1* = *lift*$_c$ *c1* **and**
    lc2: *lc2* = *lift*$_c$ *c2*
    by (*auto simp add*: *lift*$_c$-*Seq lift*$_c$-*def*)
  show *?case*
  proof (*cases s'*)
    case (*Normal s''*)
    from *Seq.hyps* (*2*) [*OF lc1* [*symmetric*]] *this*
    have *s''* = *inject s* (*project s''*)
      by *auto*
    moreover from *Seq.hyps* (*4*) [*OF lc2* [*symmetric*]] *Normal t*
    have *state t* = *inject s''* (*project* (*state t*))
      by *auto*
    ultimately have *state t* = *inject* (*inject s* (*project s''*)) (*project* (*state t*))
      by *simp*
    then show *?thesis*
      by (*simp add*: *inject-last*)
  next
    case (*Abrupt s''*)
    from *Seq.hyps* (*2*) [*OF lc1* [*symmetric*]] *this*
    have *s''* = *inject s* (*project s''*)
      by *auto*
    moreover from *Seq.hyps* (*4*) [*OF lc2* [*symmetric*]] *Abrupt t*
    have *state t* = *inject s''* (*project* (*state t*))
      by *auto*
    ultimately have *state t* = *inject* (*inject s* (*project s''*)) (*project* (*state t*))
      by *simp*
    then show *?thesis*

by (*simp add*: *inject-last*)
next
case (*Fault f*)
with *Seq*
have $t = Fault\ f$
by (*auto dest*: *Fault-end*)
with *t* **have** *False* **by** *simp*
thus *?thesis* **..**
next
case *Stuck*
with *Seq*
have $t = Stuck$
by (*auto dest*: *Stuck-end*)
with *t* **have** *False* **by** *simp*
thus *?thesis* **..**
qed
next
case *CondTrue* **thus** *?case*
by (*clarsimp simp add*: $lift_c$-*Cond* $lift_c$-*def*)
next
case *CondFalse* **thus** *?case*
by (*clarsimp simp add*: $lift_c$-*Cond* $lift_c$-*def*)
next
case ($WhileTrue\ s\ lb\ lc'\ s'\ t\ c$)
have *t*: $t \notin Fault\ `\ UNIV \cup \{Stuck\}$ **by** *fact*
have *lw*: $lift_c\ c = While\ lb\ lc'$ **by** *fact*
then obtain $b\ c'$ **where**
c: $c = While\ b\ c'$ **and**
lb: $lb = lift_s\ b$ **and**
lc: $lc' = lift_c\ c'$
by (*auto simp add*: $lift_c$-*While* $lift_s$-*def* $lift_c$-*def*)
show *?case*
proof (*cases* s')
case (*Normal* s'')
from *WhileTrue.hyps* (*3*) [*OF lc* [*symmetric*]] *this*
have $s'' = inject\ s\ (project\ s'')$
by *auto*
moreover from *WhileTrue.hyps* (*5*) [*OF lw*] *Normal t*
have $state\ t = inject\ s''\ (project\ (state\ t))$
by *auto*
ultimately have $state\ t = inject\ (inject\ s\ (project\ s''))\ (project\ (state\ t))$
by *simp*
then show *?thesis*
by (*simp add*: *inject-last*)
next
case (*Abrupt* s'')
from *WhileTrue.hyps* (*3*) [*OF lc* [*symmetric*]] *this*
have $s'' = inject\ s\ (project\ s'')$
by *auto*

moreover from *WhileTrue.hyps* (*5*) [*OF lw*] *Abrupt t*
have *state t* = *inject s''* (*project* (*state t*))
  by *auto*
ultimately have *state t* = *inject* (*inject s* (*project s''*)) (*project* (*state t*))
  by *simp*
then show *?thesis*
  by (*simp add*: *inject-last*)
next
case (*Fault f*)
with *WhileTrue*
have *t* = *Fault f*
  by (*auto dest*: *Fault-end*)
with *t* **have** *False* **by** *simp*
thus *?thesis* **..**
next
case *Stuck*
with *WhileTrue*
have *t* = *Stuck*
  by (*auto dest*: *Stuck-end*)
with *t* **have** *False* **by** *simp*
thus *?thesis* **..**
qed
next
case *WhileFalse* **thus** *?case*
  by (*clarsimp simp add*: $lift_c$*-While inj-proj-commute*)
next
case *Call* **thus** *?case*
  by (*clarsimp simp add*: *inject-last* $lift_c$*-Call* $lift_e$*-def* $lift_c$*-def*)
next
case *CallUndefined* **thus** *?case* **by** *simp*
next
case *StuckProp* **thus** *?case* **by** *simp*
next
case *DynCom*
thus *?case*
  by (*clarsimp simp add*: $lift_c$*-DynCom* $lift_c$*-def*)
next
case *Throw* **thus** *?case*
  by (*simp add*: *inj-proj-commute*)
next
case *AbruptProp* **thus** *?case* **by** (*simp add*: *inj-proj-commute*)
next
case (*CatchMatch lc1 s s' lc2 t c*)
have *t*: $t \notin$ *Fault* ‘ *UNIV* $\cup$ {*Stuck*} **by** *fact*
have $lift_c$ *c* = *Catch lc1 lc2* **by** *fact*
then obtain *c1 c2* **where**
  c: *c* = *Catch c1 c2* **and**
  lc1: *lc1* = $lift_c$ *c1* **and**
  lc2: *lc2* = $lift_c$ *c2*

by (*auto simp add*: $lift_c$*-Catch* $lift_c$*-def*)
from *CatchMatch.hyps* (*2*) [*OF lc1* [*symmetric*]] *this*
have $s' = inject\ s\ (project\ s')$
by *auto*
moreover
from *CatchMatch.hyps* (*4*) [*OF lc2* [*symmetric*]] *t*
have $state\ t = inject\ s'\ (project\ (state\ t))$
by *auto*
ultimately have $state\ t = inject\ (inject\ s\ (project\ s'))\ (project\ (state\ t))$
by *simp*
then show *?case*
by (*simp add*: *inject-last*)
next
case *CatchMiss*
thus *?case*
by (*clarsimp simp add*: $lift_c$*-Catch* $lift_c$*-def*)
qed

lemma (**in** *lift-state-space-ext*) *valid-inject-project*:
assumes *noFaultStuck*:
$\Gamma\vdash\langle c, Normal\ (project\ \sigma)\rangle \Rightarrow\notin(Fault\ `\ UNIV \cup \{Stuck\})$
shows $lift_e\ \Gamma\models_{/F}\ \{\sigma\}\ lift_c\ c$
$\{t.\ t{=}inject\ \sigma\ (project\ t)\}, \{t.\ t{=}inject\ \sigma\ (project\ t)\}$
proof (*rule validI*)
fix *s t*
assume *exec*: $lift_e\ \Gamma\vdash\langle lift_c\ c, Normal\ s\rangle \Rightarrow t$
assume *P*: $s \in \{\sigma\}$
assume *noFault*: $t \notin Fault\ `\ F$
show $t \in Normal\ `\ \{t.\ t = inject\ \sigma\ (project\ t)\} \cup$
$Abrupt\ `\ \{t.\ t = inject\ \sigma\ (project\ t)\}$
proof −
from *lift-exec* [*OF exec*]
have $\Gamma\vdash\langle c, project_x\ (Normal\ s)\rangle \Rightarrow project_x\ t$
by *simp*
with *noFaultStuck P* **have** *t*: $t \notin Fault\ `\ UNIV \cup \{Stuck\}$
by (*auto simp add*: *final-notin-def* $project_x$*-def*)
from *lift-exec-inject-same* [*OF exec refl this*] *P*
have $state\ t = inject\ \sigma\ (project\ (state\ t))$
by *simp*
with *t* **show** *?thesis*
by (*cases t*) *auto*
qed
qed

lemma (**in** *lift-state-space-ext*) *lift-exec-inject-same'*:
assumes *exec-lc*: $(lift_e\ \Gamma)\vdash\langle lift_c\ c, S\rangle \Rightarrow T$
shows $\bigwedge c.\ [\![T \notin (Fault\ `\ UNIV) \cup \{Stuck\}]\!] \Longrightarrow$
$state\ T = inject\ (state\ S)\ (project\ (state\ T))$
using *lift-exec-inject-same* [*OF exec-lc*]

by *simp*

lemma (**in** *lift-state-space-ext*) *valid-lift-modifies*:
assumes *valid*: $\forall s.\ \Gamma\models_{/F} \{s\}\ c\ (\mathit{Modif}\ s),(\mathit{ModifAbr}\ s)$
shows $(\mathit{lift}_e\ \Gamma)\models_{/F} \{S\}\ (\mathit{lift}_c\ c)$
$\{T.\ T \in \mathit{lift}_s\ (\mathit{Modif}\ (\mathit{project}\ S)) \wedge T{=}\mathit{inject}\ S\ (\mathit{project}\ T)\}$,
$\{T.\ T \in \mathit{lift}_s\ (\mathit{ModifAbr}\ (\mathit{project}\ S)) \wedge T{=}\mathit{inject}\ S\ (\mathit{project}\ T)\}$
proof (*rule validI*)
fix *s t*
assume *exec*: $\mathit{lift}_e\ \Gamma\vdash\langle \mathit{lift}_c\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
assume *P*: $s \in \{S\}$
assume *noFault*: $t \notin \mathit{Fault}$ ‘ F
show $t \in \mathit{Normal}$ ‘
$\{t \in \mathit{lift}_s\ (\mathit{Modif}\ (\mathit{project}\ S)).$
$t = \mathit{inject}\ S\ (\mathit{project}\ t)\} \cup$
Abrupt ‘
$\{t \in \mathit{lift}_s\ (\mathit{ModifAbr}\ (\mathit{project}\ S)).$
$t = \mathit{inject}\ S\ (\mathit{project}\ t)\}$
proof −
from *lift-exec* [*OF exec*]
have $\Gamma\vdash \langle c,\mathit{project}_x\ (\mathit{Normal}\ s)\rangle \Rightarrow \mathit{project}_x\ t$
by *auto*
moreover
from *noFault* **have** $\mathit{project}_x\ t \notin \mathit{Fault}$ ‘ F
by (*cases t*) (*auto simp add*: $\mathit{project}_x$*-def*)
ultimately
have $\mathit{project}_x\ t \in$
Normal ‘ (*Modif* (*project s*)) ∪ *Abrupt* ‘ (*ModifAbr* (*project s*))
using *valid* [*rule-format, of* (*project s*)]
by (*auto simp add*: *valid-def* $\mathit{project}_x$*-def*)
hence *t*: $t \in \mathit{Normal}$ ‘ lift_s (*Modif* (*project s*)) ∪
Abrupt ‘ lift_s (*ModifAbr* (*project s*))
by (*cases t*) (*auto simp add*: $\mathit{project}_x$*-def* lift_s*-def Compose.*lift_s*-def*)
then have $t \notin \mathit{Fault}$ ‘ *UNIV* ∪ {*Stuck*}
by (*cases t*) *auto*
from *lift-exec-inject-same* [*OF exec - this*]
have *state t* = *inject* (*state* (*Normal s*)) (*project* (*state t*))
by *simp*
with *t* **show** *?thesis*
using *P* **by** *auto*
qed
qed

lemma (**in** *lift-state-space-ext*) *hoare-lift-modifies*:
assumes *deriv*: $\forall \sigma.\ \Gamma,\{\}\vdash_{/F} \{\sigma\}\ c\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
shows $\forall \sigma.\ (\mathit{lift}_e\ \Gamma),\{\}\vdash_{/F} \{\sigma\}\ (\mathit{lift}_c\ c)$
$\{T.\ T \in \mathit{lift}_s\ (\mathit{Modif}\ (\mathit{project}\ \sigma)) \wedge T{=}\mathit{inject}\ \sigma\ (\mathit{project}\ T)\}$,
$\{T.\ T \in \mathit{lift}_s\ (\mathit{ModifAbr}\ (\mathit{project}\ \sigma)) \wedge T{=}\mathit{inject}\ \sigma\ (\mathit{project}\ T)\}$
apply (*rule allI*)

apply (*rule hoare-complete*)
apply (*rule valid-lift-modifies*)
apply (*rule allI*)
apply (*insert hoare-sound* [*OF deriv* [*rule-format*]])
apply (*simp add*: *cvalid-def*)
done

lemma (**in** *lift-state-space-ext*) *hoare-lift-modifies'*:
  assumes *deriv*: $\forall \sigma.\ \Gamma,\{\}\vdash_{/F} \{\sigma\}\ c\ (Modif\ \sigma),(ModifAbr\ \sigma)$
  shows $\forall \sigma.\ (lift_e\ \Gamma),\{\}\vdash_{/F} \{\sigma\}\ (lift_c\ c)$
  $\{T.\ T \in lift_s\ (Modif\ (project\ \sigma)) \wedge$
    $(\exists T'.\ T{=}inject\ \sigma\ T')\}$,
  $\{T.\ T \in lift_s\ (ModifAbr\ (project\ \sigma)) \wedge$
    $(\exists T'.\ T{=}inject\ \sigma\ T')\}$
apply (*rule allI*)
apply (*rule HoarePartialDef.conseq* [*OF hoare-lift-modifies* [*OF deriv*]])
apply *blast*
done

25.2 Renaming Procedures

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

lemma *rename-Skip*: *rename h c = Skip* = (*c=Skip*)
  by (*cases c*) *auto*

lemma *rename-Basic*:
  (*rename h c = Basic f*) = (*c=Basic f*)
  by (*cases c*) *auto*

lemma *rename-Spec*:
  (*rename h c = Spec r*) = (*c=Spec r*)
  by (*cases c*) *auto*

lemma *rename-Seq*:
  (*rename h c = Seq* rc_1 rc_2) =
    ($\exists$ c_1 c_2. c = *Seq* c_1 c_2 $\wedge$

$rc_1 = rename\ h\ c_1 \wedge rc_2 = rename\ h\ c_2$)
by (*cases c*) *auto*

lemma *rename-Cond*:
($rename\ h\ c = Cond\ b\ rc_1\ rc_2$) =
($\exists c_1\ c_2.\ c = Cond\ b\ c_1\ c_2 \wedge rc_1 = rename\ h\ c_1 \wedge rc_2 = rename\ h\ c_2$)
by (*cases c*) *auto*

lemma *rename-While*:
($rename\ h\ c = While\ b\ rc'$) = ($\exists c'.\ c = While\ b\ c' \wedge rc' = rename\ h\ c'$)
by (*cases c*) *auto*

lemma *rename-Call*:
($rename\ h\ c = Call\ q$) = ($\exists p.\ c = Call\ p \wedge q=h\ p$)
by (*cases c*) *auto*

lemma *rename-DynCom*:
($rename\ h\ c = DynCom\ rc$) = ($\exists C.\ c=DynCom\ C \wedge rc = (\lambda s.\ rename\ h\ (C\ s))$)
by (*cases c*) *auto*

lemma *rename-Guard*:
($rename\ h\ c = Guard\ f\ g\ rc'$) =
($\exists c'.\ c = Guard\ f\ g\ c' \wedge rc' = rename\ h\ c'$)
by (*cases c*) *auto*

lemma *rename-Throw*:
($rename\ h\ c = Throw$) = ($c = Throw$)
by (*cases c*) *auto*

lemma *rename-Catch*:
($rename\ h\ c = Catch\ rc_1\ rc_2$) =
($\exists c_1\ c_2.\ c = Catch\ c_1\ c_2 \wedge rc_1 = rename\ h\ c_1 \wedge rc_2 = rename\ h\ c_2$)
by (*cases c*) *auto*

lemma *exec-rename-to-exec*:
assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (h\ p) = Some\ (rename\ h\ bdy)$
assumes *exec*: $\Gamma' \vdash \langle rc,s \rangle \Rightarrow t$
shows $\bigwedge c.\ rename\ h\ c = rc \Longrightarrow\ \exists t'.\ \Gamma \vdash \langle c,s \rangle \Rightarrow t' \wedge (t'=Stuck \vee t'=t)$
using *exec*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Skip*)
next
case *Guard* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Guard*)
next
case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Guard*)
next
case *FaultProp* **thus** *?case* **by** (*fastforce intro*: *exec.intros*)
next

case *Basic* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Basic*)
next
case *Spec* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Spec*)
next
case *SpecStuck* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Spec*)
next
case *Seq* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Seq*)
next
case *CondTrue* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Cond*)
next
case *CondFalse* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Cond*)
next
case *WhileTrue* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-While*)
next
case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-While*)
next
case (*Call p rbdy s t*)
have *rbdy*: Γ' *p* = *Some rbdy* **by** *fact*
have *rename h c* = *Call p* **by** *fact*
then obtain *q* **where** *c*: *c*=*Call q* **and** *p*: *p*=*h q*
by (*auto simp add*: *rename-Call*)
show *?case*
proof (*cases* Γ *q*)
case *None*
with *c* **show** *?thesis* **by** (*auto intro*: *exec.CallUndefined*)
next
case (*Some bdy*)
from Γ [*rule-format*, *OF this*] *p rbdy*
have *rename h bdy* = *rbdy* **by** *simp*
with *Call.hyps c Some*
show *?thesis*
by (*fastforce intro*: *exec.intros*)
qed
next
case (*CallUndefined p s*)
have *undef*: Γ' *p* = *None* **by** *fact*
have *rename h c* = *Call p* **by** *fact*
then obtain *q* **where** *c*: *c*=*Call q* **and** *p*: *p*=*h q*
by (*auto simp add*: *rename-Call*)
from *undef p* Γ **have** Γ *q* = *None*
by (*cases* Γ *q*) *auto*
with *p c* **show** *?case*
by (*auto intro*: *exec.intros*)
next
case *StuckProp* **thus** *?case* **by** (*fastforce intro*: *exec.intros*)
next
case *DynCom* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-DynCom*)
next
case *Throw* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Throw*)

next
  case *AbruptProp* **thus** *?case* **by** (*fastforce intro*: *exec.intros*)
next
  case *CatchMatch* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Catch*)
next
  case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *exec.intros simp add*: *rename-Catch*)
qed

lemma *exec-rename-to-exec'*:
  assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
  assumes *exec*: $\Gamma'\vdash\langle rename\ N\ c,s\rangle \Rightarrow t$
  shows $\exists t'.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t' \wedge (t'=Stuck \vee t'=t)$
  using *exec-rename-to-exec* [*OF* Γ *exec*]
  by *auto*

lemma *valid-to-valid-rename*:
  assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
  assumes *valid*: $\Gamma\models_{/F}\ P\ c\ Q,A$
  shows $\Gamma'\models_{/F}\ P\ (rename\ N\ c)\ Q,A$
proof (*rule validI*)
  fix *s t*
  assume *execr*: $\Gamma'\vdash\ \langle rename\ N\ c,Normal\ s\rangle \Rightarrow t$
  assume *P*: $s \in P$
  assume *noFault*: $t \notin Fault\ `\ F$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof −
    from *exec-rename-to-exec* [*OF* Γ *execr*]
    obtain t' **where**
      exec: $\Gamma\vdash\ \langle c,Normal\ s\rangle \Rightarrow t'$ **and** t': $(t' = Stuck \vee t' = t)$
      by *auto*
    with *valid noFault P* **show** *?thesis*
      by (*auto simp add*: *valid-def*)
  qed
qed

lemma *hoare-to-hoare-rename*:
  assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
  assumes *deriv*: $\Gamma,\{\}\vdash_{/F}\ P\ c\ Q,A$
  shows $\Gamma',\{\}\vdash_{/F}\ P\ (rename\ N\ c)\ Q,A$
apply (*rule hoare-complete*)
apply (*insert hoare-sound* [*OF deriv*])
apply (*rule valid-to-valid-rename*)
apply (*rule* Γ)
apply (*simp add*: *cvalid-def*)
done

lemma *hoare-to-hoare-rename'*:
  assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
  assumes *deriv*: $\forall Z.\ \Gamma,\{\}\vdash_{/F}\ (P\ Z)\ c\ (Q\ Z),(A\ Z)$
  shows $\forall Z.\ \Gamma',\{\}\vdash_{/F}\ (P\ Z)\ (rename\ N\ c)\ (Q\ Z),(A\ Z)$
apply *rule*
apply (*rule hoare-to-hoare-rename* [*OF* Γ])
apply (*rule deriv*[*rule-format*])
done

lemma *terminates-to-terminates-rename*:
  assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
  assumes *termi*: $\Gamma\vdash\ c \downarrow s$
  assumes *noStuck*: $\Gamma\vdash\ \langle c,s\rangle \Rightarrow\notin\{Stuck\}$
  shows $\Gamma'\vdash\ rename\ N\ c \downarrow s$
using *termi noStuck*
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*
    simp add: *final-notin-def exec.intros*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Fault* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
  case *Seq*
  thus *?case*
    by (*force intro*!: *terminates.intros exec.intros dest*: *exec-rename-to-exec* [*OF* Γ]
        simp add: *final-notin-def*)
next
  case *CondTrue* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*
    simp add: *final-notin-def exec.intros*)
next
  case *CondFalse* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*
    simp add: *final-notin-def exec.intros*)
next
  case (*WhileTrue s b c*)
  have *s-in-b*: $s \in b$ **by** *fact*
  have *noStuck*: $\Gamma\vdash\ \langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
  with *s-in-b* **have** $\Gamma\vdash\ \langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
    by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
  with *WhileTrue.hyps* **have** $\Gamma'\vdash rename\ N\ c \downarrow Normal\ s$
    by *simp*
  moreover

{
fix t
assume *exec-rc*: $\Gamma'\vdash \langle rename\ N\ c,Normal\ s\rangle \Rightarrow t$
have $\Gamma'\vdash$ *While b* (*rename N c*) $\downarrow t$
proof −
from *exec-rename-to-exec* [*OF* Γ *exec-rc*] **obtain** t'
where *exec-c*: $\Gamma\vdash \langle c,Normal\ s\rangle \Rightarrow t'$ **and** t': $(t' = Stuck \vee t' = t)$
by *auto*
with *s-in-b noStuck* **obtain** $t'=t$ **and** $\Gamma\vdash \langle While\ b\ c,t\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
with *exec-c WhileTrue.hyps*
show *?thesis*
by *auto*
qed
}
ultimately show *?case*
using *s-in-b*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case (*Call p bdy s*)
have Γ p = *Some bdy* **by** *fact*
from Γ [*rule-format*, *OF this*]
have *bdy'*: Γ' (N p) = *Some* (*rename N bdy*).
from *Call* **have** $\Gamma'\vdash$*rename N bdy* $\downarrow$ *Normal s*
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
with *bdy'* **have** $\Gamma'\vdash$*Call* (N p) $\downarrow$ *Normal s*
by (*auto intro*: *terminates.intros*)
thus *?case* **by** *simp*
next
case (*CallUndefined p s*)
have Γ p = *None* $\Gamma\vdash \langle Call\ p,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$ **by** *fact+*
hence *False* **by** (*auto simp add*: *final-notin-def intro*: *exec.intros*)
thus *?case* **..**
next
case *Stuck* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case* **by** (*fastforce intro*: *terminates.intros*
simp add: *final-notin-def exec.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 *noStuck*: $\Gamma\vdash \langle Catch\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
hence $\Gamma\vdash \langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)

with *Catch.hyps* **have** $\Gamma'\vdash$*rename N c1* $\downarrow$ *Normal s*
  by *auto*
moreover
{
  fix *t*
  assume *exec-rc1*:$\Gamma'\vdash$ $\langle$*rename N c1*,*Normal s*$\rangle$ $\Rightarrow$ *Abrupt t*
  have $\Gamma'\vdash$*rename N c2* $\downarrow$ *Normal t*
  proof $-$
    from *exec-rename-to-exec* [*OF* Γ *exec-rc1*] **obtain** t'
      where *exec-c*: $\Gamma\vdash$ $\langle$*c1*,*Normal s*$\rangle$ $\Rightarrow$ t' **and** (t' = *Stuck* $\vee$ t' = *Abrupt t*)
      by *auto*
    with *noStuck* **have** t': t'=*Abrupt t*
      by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
    with *exec-c noStuck* **have** $\Gamma\vdash$ $\langle$*c2*,*Normal t*$\rangle$ $\Rightarrow\notin$\{*Stuck*\}
      by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
    with *exec-c* t' *Catch.hyps*
    show *?thesis*
      by *auto*
  qed
}
ultimately show *?case*
  by (*auto intro*: *terminates.intros*)
qed

lemma *validt-to-validt-rename*:
  assumes Γ: $\forall$ *p bdy*. Γ *p* = *Some bdy* $\longrightarrow$ Γ' (*N p*) = *Some* (*rename N bdy*)
  assumes *valid*: $\Gamma\models_{t/F}$ *P c Q*,*A*
  shows $\Gamma'\models_{t/F}$ *P* (*rename N c*) *Q*,*A*
proof $-$
  from *valid*
  have $\Gamma'\models_{/F}$ *P* (*rename N c*) *Q*,*A*
    by (*auto intro*: *valid-to-valid-rename* [*OF* Γ] *simp add*: *validt-def*)
  moreover
  {
    fix *s*
    assume *s* $\in$ *P*
    with *valid* **obtain** $\Gamma\vdash$*c* $\downarrow$ (*Normal s*) $\Gamma\vdash$ $\langle$*c*,*Normal s*$\rangle$ $\Rightarrow\notin$\{*Stuck*\}
      by (*auto simp add*: *validt-def valid-def final-notin-def*)
    from *terminates-to-terminates-rename* [*OF* Γ *this*]
    have $\Gamma'\vdash$*rename N c* $\downarrow$ *Normal s*
      .
  }
  ultimately show *?thesis*
    by (*simp add*: *validt-def*)
qed

lemma *hoaret-to-hoaret-rename*:
  assumes Γ: $\forall$ *p bdy*. Γ *p* = *Some bdy* $\longrightarrow$ Γ' (*N p*) = *Some* (*rename N bdy*)
  assumes *deriv*: Γ,\{\}$\vdash_{t/F}$ *P c Q*,*A*

shows $\Gamma',\{\}\vdash_{t/F}$ *P* (*rename N c*) *Q*,*A*
apply (*rule hoaret-complete*)
apply (*insert hoaret-sound* [*OF deriv*])
apply (*rule validt-to-validt-rename*)
apply (*rule* Γ)
apply (*simp add*: *cvalidt-def*)
done

lemma *hoaret-to-hoaret-rename'*:
assumes Γ: $\forall p\ bdy.\ \Gamma\ p = Some\ bdy \longrightarrow \Gamma'\ (N\ p) = Some\ (rename\ N\ bdy)$
assumes *deriv*: $\forall Z.\ \Gamma,\{\}\vdash_{t/F} (P\ Z)\ c\ (Q\ Z),(A\ Z)$
shows $\forall Z.\ \Gamma',\{\}\vdash_{t/F} (P\ Z)\ (rename\ N\ c)\ (Q\ Z),(A\ Z)$
apply *rule*
apply (*rule hoaret-to-hoaret-rename* [*OF* Γ])
apply (*rule deriv*[*rule-format*])
done

lemma $lift_c$-*whileAnno* [*simp*]: $lift_c$ *prj inject* (*whileAnno b I V c*) =
whileAnno ($lift_s$ *prj b*)
($lift_s$ *prj I*) ($lift_r$ *prj inject V*) ($lift_c$ *prj inject c*)
by (*simp add*: *whileAnno-def*)

lemma $lift_c$-*block* [*simp*]: $lift_c$ *prj inject* (*block init bdy return c*) =
block ($lift_f$ *prj inject init*) ($lift_c$ *prj inject bdy*)
(λ*s*. ($lift_f$ *prj inject* (*return* (*prj s*))))
(λ*s t*. $lift_c$ *prj inject* (*c* (*prj s*) (*prj t*)))
by (*simp add*: *block-def*)

lemma $lift_c$-*call* [*simp*]: $lift_c$ *prj inject* (*call init p return c*) =
call ($lift_f$ *prj inject init*) *p*
(λ*s*. ($lift_f$ *prj inject* (*return* (*prj s*))))
(λ*s t*. $lift_c$ *prj inject* (*c* (*prj s*) (*prj t*)))
by (*simp add*: *call-def* $lift_c$-*block*)

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

lemma *rename-block* [*simp*]: *rename h* (*block init bdy return c*) =
block init (*rename h bdy*) *return* (λ*s t*. *rename h* (*c s t*))
by (*simp add*: *block-def*)

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

end

theory *ComposeEx* **imports** *Compose ../Vcg ../HeapList* **begin**

record *globals-list* =
  next-' :: *ref* ⇒ *ref*

record *state-list* = *globals-list state* +
  p-'   :: *ref*
  sl-q-'   :: *ref*
  r-'   :: *ref*

procedures *Rev*(*p*|*sl-q*) =
      ´*sl-q* :== *Null*;;
      WHILE ´*p* ≠ *Null*
      DO
        ´*r* :== ´*p*;; {|´*p* ≠ *Null*|}⟼ ´*p* :== ´*p*→´*next*;;
        {|´*r* ≠ *Null*|}⟼ ´*r*→´*next* :== ´*sl-q*;; ´*sl-q* :== ´*r*
      OD
print-theorems

lemma (**in** *Rev-impl*)
 Rev-modifies:
  ∀σ. Γ⊢$_{/UNIV}$ {σ} ´*sl-q* :== *PROC Rev*(´*p*) {*t*. *t may-only-modify-globals* σ *in* [*next*]}
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*vcg spec=modifies*)
done

lemma (**in** *Rev-impl*) **shows**
 Rev-spec:
  ∀ *Ps*. Γ⊢ {|*List* ´*p* ´*next Ps*|} ´*sl-q* :== *PROC Rev*(´*p*) {|*List* ´*sl-q* ´*next* (*rev Ps*)|}
apply (*hoare-rule HoarePartial.ProcNoRec1*)
apply (*hoare-rule anno* =
      ´*sl-q* :== *Null*;;
      WHILE ´*p* ≠ *Null INV* {|∃ *Ps' Qs'*. *List* ´*p* ´*next Ps'* ∧ *List* ´*sl-q* ´*next Qs'* ∧
                    set Ps' ∩ *set Qs'* = {} ∧
                    rev Ps' @ *Qs'* = *rev Ps*|}
       DO
        ´*r* :== ´*p*;; {|´*p* ≠ *Null*|}⟼´*p* :== ´*p*→´*next*;;
        {|´*r* ≠ *Null*|}⟼ ´*r*→´*next* :== ´*sl-q*;; ´*sl-q* :== ´*r*
      OD **in** *HoarePartial.annotateI*)
apply *vcg*
apply *clarsimp*

apply *fastforce*
apply *clarsimp*
done

declare $[[names\text{-}unique = false]]$

record *globals* =
  $strnext\text{-}'$ :: $ref \Rightarrow ref$
  $chr\text{-}'$ :: $ref \Rightarrow char$

  $qnext\text{-}'$ :: $ref \Rightarrow ref$
  $cont\text{-}'$ :: $ref \Rightarrow int$
record *state* = *globals state* +
  $str\text{-}'$ :: *ref*
  $queue\text{-}'$:: *ref*
  $q\text{-}'$ :: *ref*
  $r\text{-}'$ :: *ref*

definition *project-globals-str*:: *globals* ⇒ *globals-list*
  where *project-globals-str* g = $(\!|next\text{-}' = strnext\text{-}'\ g|\!)$

definition *project-str*:: *state* ⇒ *state-list*
where
project-str s =
  $(\!|globals = project\text{-}globals\text{-}str\ (globals\ s),$
   $state\text{-}list.p\text{-}' = str\text{-}'\ s,\ sl\text{-}q\text{-}' = q\text{-}'\ s,\ state\text{-}list.r\text{-}' = r\text{-}'\ s|\!)$

definition *inject-globals-str*::
  globals ⇒ *globals-list* ⇒ *globals*
where
  inject-globals-str $G\ g$ =
   $G(\!|strnext\text{-}' := next\text{-}'\ g|\!)$

definition *inject-str*::*state* ⇒ *state-list* ⇒ *state* **where**
inject-str $S\ s = S(\!|globals := inject\text{-}globals\text{-}str\ (globals\ S)\ (globals\ s),$
   $str\text{-}' := state\text{-}list.p\text{-}'\ s,\ q\text{-}' := sl\text{-}q\text{-}'\ s,$
   $r\text{-}' := state\text{-}list.r\text{-}'\ s|\!)$

lemma *globals-inject-project-str-commutes*:
  inject-globals-str G (*project-globals-str* G) = G
  by (*simp add*: *inject-globals-str-def project-globals-str-def*)

lemma *inject-project-str-commutes*: *inject-str* S (*project-str* S) = S
  by (*simp add*: *inject-str-def project-str-def globals-inject-project-str-commutes*)

lemma *globals-project-inject-str-commutes*:
  project-globals-str (*inject-globals-str* $G\ g$) = g
  by (*simp add*: *inject-globals-str-def project-globals-str-def*)

lemma *project-inject-str-commutes*: *project-str* (*inject-str* S s) = s
  by (*simp add*: *inject-str-def project-str-def globals-project-inject-str-commutes*)

lemma *globals-inject-str-last*:
  inject-globals-str (*inject-globals-str* G g) g' = *inject-globals-str* G g'
  by (*simp add*: *inject-globals-str-def*)

lemma *inject-str-last*:
  inject-str (*inject-str* S s) s' = *inject-str* S s'
  by (*simp add*: *inject-str-def globals-inject-str-last*)

definition
  $lift_e$ = ($\lambda\Gamma$ p. *map-option* ($lift_c$ *project-str inject-str*) (Γ p))
print-locale *lift-state-space*
interpretation *ex*: *lift-state-space project-str inject-str*
  xstate-map project-str $lift_e$ $lift_c$ *project-str inject-str*
  $lift_f$ *project-str inject-str* $lift_s$ *project-str*
  $lift_r$ *project-str inject-str*
  apply −
  apply (*rule lift-state-space.intro*)
  apply (*rule project-inject-str-commutes*)
  apply *simp*
  apply *simp*
  apply (*simp add*: $lift_e$-*def*)
  apply *simp*
  apply *simp*
  apply *simp*
  done

interpretation *ex*: *lift-state-space-ext project-str inject-str*
  xstate-map project-str $lift_e$ $lift_c$ *project-str inject-str*
  $lift_f$ *project-str inject-str* $lift_s$ *project-str*
  $lift_r$ *project-str inject-str*

apply −
apply *intro-locales* [*1*]
  apply (*rule lift-state-space-ext-axioms.intro*)
  apply (*rule inject-project-str-commutes*)
  apply (*rule inject-str-last*)
apply (*simp-all add*: $lift_e$-*def*)
  done

lemmas *Rev-lift-spec* = *ex.lift-hoarep'* [*OF Rev-impl.Rev-spec*,*simplified* $lift_s$-*def*
 project-str-def project-globals-str-def,*simplified*, *of* - ''*Rev*'']

print-theorems

definition $\mathcal{N}\ p'\ p = (if\ p{=}''Rev''\ then\ p'\ else\ '''')$

procedures *RevStr*$(str|q)$ = *rename* ($\mathcal{N}$ *RevStr-'proc*)
(*lift*$_c$ *project-str inject-str* (*Rev-body.Rev-body*))

lemmas *Rev-lift-spec'* =
Rev-lift-spec [*of* [''*Rev*''↦*Rev-body.Rev-body*] ,
simplified Rev-impl-def Rev-clique-def ,*simplified*]
thm *Rev-lift-spec'*

lemma *Rev-lift-spec''*:
∀ *Ps*. *lift*$_e$ [''*Rev*'' ↦ *Rev-body.Rev-body*]
⊢ {|*List ´str ´strnext Ps*|} *Call* ''*Rev*'' {|*List ´q ´strnext* (*rev Ps*)|}
by (*rule Rev-lift-spec'*)

lemma (**in** *RevStr-impl*) $\mathcal{N}$*-ok*:
∀ *p bdy*. (*lift*$_e$ [''*Rev*'' ↦ *Rev-body.Rev-body*]) *p* = *Some bdy* ⟶
Γ ($\mathcal{N}$ *RevStr-'proc p*) = *Some* (*rename* ($\mathcal{N}$ *RevStr-'proc*) *bdy*)
apply (*insert RevStr-impl*)
apply (*auto simp add*: *RevStr-body-def lift*$_e$*-def* $\mathcal{N}$*-def*)
done

context *RevStr-impl*
begin
thm *hoare-to-hoare-rename'*[*OF - Rev-lift-spec''*, *OF* $\mathcal{N}$*-ok*,
simplified $\mathcal{N}$*-def*, *simplified*]
end

lemmas (**in** *RevStr-impl*) *RevStr-spec* =
hoare-to-hoare-rename' [*OF - Rev-lift-spec''*, *OF* $\mathcal{N}$*-ok*,
simplified $\mathcal{N}$*-def*, *simplified*]

lemma (**in** *RevStr-impl*) *RevStr-spec'*:
∀ *Ps*. Γ⊢ {|*List ´str ´strnext Ps*|} *´q* :== *PROC RevStr*(*´str*)
{|*List ´q ´strnext* (*rev Ps*)|}
by (*rule RevStr-spec*)

lemmas *Rev-modifies'* =
Rev-impl.Rev-modifies [*of* [''*Rev*''↦*Rev-body.Rev-body*], *simplified Rev-impl-def*,
simplified]
thm *Rev-modifies'*

context *RevStr-impl*
begin
lemmas *RevStr-modifies'* =
  hoare-to-hoare-rename' [*OF* - *ex.hoare-lift-modifies'* [*OF Rev-modifies'*],
      OF $\mathcal{N}$*-ok*, *of* ''*Rev*'', *simplified* $\mathcal{N}$*-def Rev-clique-def*,*simplified*]
end

lemma (**in** *RevStr-impl*) *RevStr-modifies*:
$\forall\sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}$ ´*str* :== *PROC RevStr*(´*str*)
  $\{t.$ *t may-only-modify-globals* σ *in* [*strnext*]$\}$
apply (*rule allI*)
apply (*rule HoarePartialProps.ConseqMGT* [*OF RevStr-modifies'*])
apply (*clarsimp simp add*:
  $lift_s$*-def mex-def meq-def*
  project-str-def inject-str-def project-globals-str-def inject-globals-str-def)
apply *blast*
done

end

26 User Guide

We introduce the verification environment with a couple of examples that illustrate how to use the different bits and pieces to verify programs.

26.1 Basics

First of all we have to decide how to represent the state space. There are currently two implementations. One is based on records the other one on the concept called 'statespace' that was introduced with Isabelle 2007 (see `HOL/Statespace`) . In contrast to records a 'satespace' does not define a new type, but provides a notion of state, based on locales. Logically the state is modelled as a function from (abstract) names to (abstract) values and the statespace infrastructure organises distinctness of names an projection/injection of concrete values into the abstract one. Towards the user the interface of records and statespaces is quite similar. However, statespaces offer more flexibility, inherited from the locale infrastructure, in particular multiple inheritance and renaming of components.

In this user guide we prefer statespaces, but give some comments on the usage of records in Section 26.9.

hoarestate *vars* =
  A :: *nat*
  I :: *nat*
  M :: *nat*

N :: *nat*
R :: *nat*
S :: *nat*

The command **hoarestate** is a simple preprocessor for the command **statespaces** which decorates the state components with the suffix -′, to avoid cluttering the namespace. Also note that underscores are printed as hyphens in this documentation. So what you see as A-′ in this document is actually `A_'`. Every component name becomes a fixed variable in the locale *vars* and can no longer be used for logical variables.

Lookup of a component A-′ in a state s is written as $s{\cdot}A$-′, and update with a value *term* v as $s\langle A\text{-}' := v\rangle$.

To deal with local and global variables in the context of procedures the program state is organised as a record containing the two componets *locals* and *globals*. The variables defined in hoarestate *vars* reside in the *locals* part.

Here is a first example.

lemma (**in** *vars*) $\Gamma\vdash \{\!|\,\acute{}N = 5|\!\}\ \acute{}N :== 2 * \acute{}N\ \{\!|\,\acute{}N = 10|\!\}$
  apply *vcg*

 1. $\bigwedge N.\ N = 5 \Longrightarrow 2 * N = 10$

  apply *simp*

No subgoals!

  done

We enable the locale of statespace *vars* by the `in vars` directive. The verification condition generator is invoked via the *vcg* method and leaves us with the expected subgoal that can be proved by simplification.

If we refer to components (variables) of the state-space of the program we always mark these with ´ (in assertions and also in the program itself). It is the acute-symbol and is present on most keyboards. The assertions of the Hoare tuple are ordinary Isabelle sets. As we usually want to refer to the state space in the assertions, we provide special brackets for them. They can be written as `{| |}` in ASCII or $\{\!|\ |\!\}$ with symbols. Internally, marking variables has two effects. First of all we refer to the implicit state and secondary we get rid of the suffix -′. So the assertion $\{\!|\,\acute{}N = 5|\!\}$ internally gets expanded to $\{s.\ \mathit{locals}\ s\ {\cdot}N\text{-}' = 5\}$ written in ordinary set comprehension notation of Isabelle. It describes the set of states where the N-′ component is equal to *5*. An empty context and an empty postcondition for abrupt termination can be omitted. The lemma above is a shorthand for $\Gamma,\{\}\vdash \{\!|\,\acute{}N = 5|\!\}\ \acute{}N :== 2 * \acute{}N\ \{\!|\,\acute{}N = 10|\!\},\{\}$.

We can step through verification condition generation by the method *vcg-step*.

lemma (**in** *vars*) $\Gamma,\{\}\vdash \{\!|\,{}'N = 5|\!\}\ {}'N :== 2 * {}'N\ \{\!|\,{}'N = 10|\!\}$
  apply *vcg-step*

$1.\ \{\!|\,{}'N = 5|\!\} \subseteq \{\!|2 * {}'N = 10|\!\}$

The last step of verification condition generation, transforms the inclusion of state sets to the corresponding predicate on components of the state space.

  apply *vcg-step*

$1.\ \bigwedge N.\ N = 5 \Longrightarrow 2 * N = 10$

  by *simp*

Although our assertions work semantically on the state space, stepping through verification condition generation "feels" like the expected syntactic substitutions of traditional Hoare logic. This is achieved by light simplification on the assertions calculated by the Hoare rules.

lemma (**in** *vars*) $\Gamma\vdash \{\!|\,{}'N = 5|\!\}\ {}'N :== 2 * {}'N\ \{\!|\,{}'N = 10|\!\}$
  apply (*rule HoarePartial.Basic*)

$1.\ \{\!|\,{}'N = 5|\!\} \subseteq \{s \mid s(\!|locals := locals\ s\langle N\text{-}' := 2 * (locals\ s{\cdot}N\text{-}')\rangle|\!) \in \{\!|\,{}'N = 10|\!\}\}$

  apply (*simp only*: *mem-Collect-eq*)

$1.\ \{\!|\,{}'N = 5|\!\} \subseteq \{s \mid locals\ (s(\!|locals := locals\ s\langle N\text{-}' := 2 * (locals\ s{\cdot}N\text{-}')\rangle|\!)){\cdot}N\text{-}' = 10\}$

  apply (*tactic*
    ‹*Hoare.BasicSimpTac* @{*context*} *Hoare.Function false*
      [] (*K all-tac*) *1*›)

$1.\ \{\!|\,{}'N = 5|\!\} \subseteq \{\!|2 * {}'N = 10|\!\}$

  by *simp*

The next example shows how we deal with the while loop. Note the invariant annotation.

lemma (**in** *vars*)
  $\Gamma,\{\}\vdash \{\!|\,{}'M = 0 \wedge {}'S = 0|\!\}$
    $WHILE\ {}'M \neq a$
    $INV\ \{\!|\,{}'S = {}'M * b|\!\}$
    $DO\ {}'S :== {}'S + b;;\ {}'M :== {}'M + 1\ OD$
    $\{\!|\,{}'S = a * b|\!\}$
  apply *vcg*

$1.\ \bigwedge M\ S.\ [\![M = 0;\ S = 0]\!] \Longrightarrow S = M * b$
$2.\ \bigwedge M\ S.\ [\![S = M * b;\ M \neq a]\!] \Longrightarrow S + b = (M + 1) * b$
$3.\ \bigwedge M\ S.\ [\![S = M * b;\ \neg\ M \neq a]\!] \Longrightarrow S = a * b$

The verification condition generator gives us three proof obligations, stemming from the path from the precondition to the invariant, from the invariant together with loop condition through the loop body to the invariant, and finally from the invariant together with the negated loop condition to the postcondition.

apply *auto*
done

26.2 Procedures

26.2.1 Declaration

Our first procedure is a simple square procedure. We provide the command **procedures**, to declare and define a procedure.

procedures
  Square $(N::nat|R::nat)$
  where $I::nat$ **in**
  $\acute{}R :== \acute{}N * \acute{}N$

A procedure is given by the signature of the procedure followed by the procedure body. The signature consists of the name of the procedure and a list of parameters together with their types. The parameters in front of the pipe | are value parameters and behind the pipe are the result parameters. Value parameters model call by value semantics. The value of a result parameter at the end of the procedure is passed back to the caller. Local variables follow the *where*. If there are no local variables the *where* ... *in* can be omitted. The variable I is actually unused in the body, but is used in the examples below.

The procedures command provides convenient syntax for procedure calls (that creates the proper *init*, *return* and *result* functions on the fly) and creates locales and statespaces to reason about the procedure. The purpose of locales is to set up logical contexts to support modular reasoning. Locales can be seen as freeze-dried proof contexts that get alive as you setup a new lemma or theorem ([2]). The locale the user deals with is named *Square-impl*. It defines the procedure name (internally *Square-'proc*), the procedure body (named *Square-body*) and the statespaces for parameters and local and global variables. Moreover it contains the assumption Γ *Square-'proc* = *Some Square-body*, which states that the procedure is properly defined in the procedure context.

The purpose of the locale is to give us easy means to setup the context in which we prove programs correct. In this locale the procedure context Γ is fixed. So we always use this letter for the procedure specification. This is crucial, if we prove programs under the assumption of some procedure specifications.

The procedures command generates syntax, so that we can either write *CALL Square*($\acute{}I$,$\acute{}R$) or $\acute{}I$:== *CALL Square*($\acute{}R$) for the procedure call. The internal term is the following:

$call\ (\lambda s.\ s(\!|locals := locals\ s\langle N\text{-}'Square\text{-}' := locals\ s{\cdot}I\text{-}'Square\text{-}'\rangle|\!))$
$\ Square\text{-}'proc\ (\lambda s\ t.\ s(\!|globals := globals\ t|\!))$
$\ (\lambda i\ t.\ \acute{}R :== locals\ t{\cdot}R\text{-}'Square\text{-}')$

Note the additional decoration (with the procedure name) of the parameter and local variable names.

The abstract syntax for the procedure call is *call init p return result*. The *init* function copies the values of the actual parameters to the formal parameters, the *return* function copies the global variables back (in our case there are no global variables), and the *result* function additionally copies the values of the formal result parameters to the actual locations. Actual value parameters can be all kind of expressions, since we only need their value. But result parameters must be proper "lvalues": variables (including dereferenced pointers) or array locations, since we have to assign values to them.

26.2.2 Verification

A procedure specification is an ordinary Hoare tuple. We use the parameterless call for the specification; $\acute{}R :== PROC\ Square(\acute{}N)$ is syntactic sugar for *Call Square-'proc*. This emphasises that the specification describes the internal behaviour of the procedure, whereas parameter passing corresponds to the procedure call. The following precondition fixes the current value $\acute{}N$ to the logical variable n. Universal quantification of n enables us to adapt the specification to an actual parameter. The specification is used in the rule for procedure call when we come upon a call to *Square*. Thus n plays the role of the auxiliary variable Z.

To verify the procedure we need to verify the body. We use a derived variant of the general recursion rule, tailored for non recursive procedures: *HoarePartial.ProcNoRec1*:

$$[\![\forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ the\ (\Gamma\ p)\ (Q\ Z),(A\ Z);\ p \in dom\ \Gamma]\!] \Longrightarrow \forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ Call\ p\ (Q\ Z),(A\ Z)$$

The naming convention for the rule is the following: The *1* expresses that we look at one procedure, and *NoRec* that the procedure is non recursive.

lemma (**in** *Square-impl*)
shows $\forall n.\ \Gamma\vdash\{\!|\acute{}N = n|\!\}\ \ \acute{}R :== PROC\ Square(\acute{}N)\ \{\!|\acute{}R = n * n|\!\}$

The directive *in* has the effect that the context of the locale *Square-impl* is included to the current lemma, and that the lemma is added as a fact to the locale, after it is proven. The next time locale *Square-impl* is invoked this lemma is immediately available as fact, which the verification condition generator can use.

apply (*hoare-rule HoarePartial.ProcNoRec1*)

1. $\forall n.\ \Gamma\vdash \{\!|\,\acute{}N = n|\!\}\ \acute{}R :== \acute{}N * \acute{}N\ \{\!|\,\acute{}R = n * n|\!\}$

The method *hoare-rule*, like *rule* applies a single rule, but additionally does some "obvious" steps: It solves the canonical side-conditions of various Hoare-rules and it automatically expands the procedure body: With *Square-impl*: Γ *Square-'proc* = *Some Square-body* we get the procedure body out of the procedure context Γ; with *Square-body-def*: *Square-body* $\equiv$ $\acute{}R :== \acute{}N * \acute{}N$ we can unfold the definition of the body.
The proof is finished by the vcg and simp.

1. $\forall n.\ \Gamma\vdash \{\!|\,\acute{}N = n|\!\}\ \acute{}R :== \acute{}N * \acute{}N\ \{\!|\,\acute{}R = n * n|\!\}$

by *vcg simp*

If the procedure is non recursive and there is no specification given, the verification condition generator automatically expands the body.

lemma (**in** *Square-impl*) *Square-spec*:
shows $\forall n.\ \Gamma\vdash\{\!|\,\acute{}N = n|\!\}\ \ \acute{}R :== PROC\ Square(\acute{}N)\ \{\!|\,\acute{}R = n * n|\!\}$
by *vcg simp*

An important naming convention is to name the specification as *<procedure−name>-spec*. The verification condition generator refers to this name in order to search for a specification in the theorem database.

26.2.3 Usage

Let us see how we can use procedure specifications.

lemma (**in** *Square-impl*)
shows $\Gamma\vdash\{\!|\,\acute{}I = 2|\!\}\ \acute{}R :== CALL\ Square(\acute{}I)\ \{\!|\,\acute{}R = 4|\!\}$

Remember that we have already proven *Square-spec* in the locale *Square-impl*. This is crucial for verification condition generation. When reaching a procedure call, it looks for the specification (by its name) and applies the rule *HoarePartial.ProcSpec* instantiated with the specification (as last premise). Before we apply the verification condition generator, let us take some time to think of what we can expect. Let's look at the specification *Square-spec* again:

$\forall n.\ \Gamma\vdash \{\!|\,\acute{}N = n|\!\}\ \acute{}R :== PROC\ Square(\acute{}N)\ \{\!|\,\acute{}R = n * n|\!\}$

The specification talks about the formal parameters N and R. The precondition $\{\!|\,\acute{}N = n|\!\}$ just fixes the initial value of N. The actual parameters are I and R. We have to adapt the specification to this calling context. $\forall n.\ \Gamma\vdash \{\!|\,\acute{}I = n|\!\}\ \acute{}R :== CALL\ Square(\acute{}I)\ \{\!|\,\acute{}R = n * n|\!\}$. From the postcondition $\{\!|\,\acute{}R = n * n|\!\}$ we have to derive the actual postcondition $\{\!|\,\acute{}R = 4|\!\}$. So we gain something like: $\{\!|n * n = 4|\!\}$. The precondition is $\{\!|\,\acute{}I = 2|\!\}$ and the specification tells us $\{\!|\,\acute{}I = n|\!\}$ for the pre-state. So the value of n is the value of I in the pre-state. So we arrive at $\{\!|\,\acute{}I = 2|\!\} \subseteq \{\!|\,\acute{}I * \acute{}I = 4|\!\}$.

apply *vcg-step*

$1.\ \{\!|\ {}^\acute{}I = 2 |\!\} \subseteq \{\!|\forall t.\ {}^tR = {}^\acute{}I * {}^\acute{}I \longrightarrow {}^\acute{}I * {}^\acute{}I = 4 |\!\}$

The second set looks slightly more involved: $\{\!|\forall t.\ {}^tR = {}^\acute{}I * {}^\acute{}I \longrightarrow {}^\acute{}I * {}^\acute{}I = 4 |\!\}$, this is an artefact from the procedure call rule. Originally ${}^\acute{}I * {}^\acute{}I = 4$ was ${}^tR = 4$. Where t denotes the final state of the procedure and the superscript notation allows to select a component from a particular state.

apply *vcg-step*

$1.\ \bigwedge I.\ I = 2 \Longrightarrow \forall R.\ R = I * I \longrightarrow I * I = 4$

by *simp*

The adaption of the procedure specification to the actual calling context is done due to the *init*, *return* and *result* functions in the rule *HoarePartial.ProcSpec* (or in the variant *HoarePartial.ProcSpecNoAbrupt* which already incorporates the fact that the postcondition for abrupt termination is the empty set). For the readers interested in the internals, here a version without vcg.

lemma (**in** *Square-impl*)
shows $\Gamma\vdash \{\!|\ {}^\acute{}I = 2 |\!\}\ {}^\acute{}R :== \mathit{CALL}\ \mathit{Square}({}^\acute{}I)\ \{\!|\ {}^\acute{}R = 4 |\!\}$
apply (*rule HoarePartial.ProcSpecNoAbrupt* [*OF - - Square-spec*])

$1.\ \{\!|\ {}^\acute{}I = 2 |\!\}$
$\quad \subseteq \{s \mid \exists Z.\ s(\!|\mathit{locals} := \mathit{locals}\ s\langle N\text{-}'\mathit{Square}\text{-}' := \mathit{locals}\ s{\cdot}I\text{-}'\mathit{Square}\text{-}'\rangle|\!)$
$\qquad\qquad \in \{\!|\ {}^\acute{}N = Z |\!\} \wedge$
$\qquad\qquad (\forall t.\ t \in \{\!|\ {}^\acute{}R = Z * Z |\!\} \longrightarrow s(\!|\mathit{globals} := \mathit{globals}\ t|\!) \in\ ?R\ s\ t)\}$
$2.\ \forall s\ t.\ \Gamma\vdash (?R\ s\ t)\ {}^\acute{}R :== \mathit{locals}\ t{\cdot}R\text{-}'\mathit{Square}\text{-}'\ \{\!|\ {}^\acute{}R = 4 |\!\}$

This is the raw verification condition, It is interesting to see how the auxiliary variable Z is actually used. It is unified with n of the specification and fixes the state after parameter passing.

apply *simp*

$1.\ \{\!|\ {}^\acute{}I = 2 |\!\}$
$\quad \subseteq \{s \mid \forall t.\ \mathit{locals}\ t{\cdot}R\text{-}'\mathit{Square}\text{-}' =$
$\qquad\qquad (\mathit{locals}\ s{\cdot}I\text{-}'\mathit{Square}\text{-}') * (\mathit{locals}\ s{\cdot}I\text{-}'\mathit{Square}\text{-}') \longrightarrow$
$\qquad\qquad s(\!|\mathit{globals} := \mathit{globals}\ t|\!) \in\ ?R\ s\ t\}$
$2.\ \forall s\ t.\ \Gamma\vdash (?R\ s\ t)\ {}^\acute{}R :== \mathit{locals}\ t{\cdot}R\text{-}'\mathit{Square}\text{-}'\ \{\!|\ {}^\acute{}R = 4 |\!\}$

prefer *2*
apply *vcg-step*

1. $\{\!\!|\ ´I = 2\ |\!\!\}$
$\subseteq \{s \mid \forall t.\ locals\ t{\cdot}R\text{-}'Square\text{-}' =$
$(locals\ s{\cdot}I\text{-}'Square\text{-}') * (locals\ s{\cdot}I\text{-}'Square\text{-}') \longrightarrow$
$s(\!|globals := globals\ t|\!) \in \{\!\!|\ ^tR = 4\ |\!\!\}\}$

apply (*auto intro*: *ext*)
done

26.2.4 Recursion

We want to define a procedure for the factorial. We first define a HOL function that calculates it, to specify the procedure later on.

primrec *fac*:: *nat* ⇒ *nat*
where
fac 0 = 1 |
fac (Suc n) = (Suc n) ∗ fac n

Now we define the procedure.

procedures
Fac (*N*::*nat* | *R*::*nat*)
IF ´N = 0 THEN ´R :== 1
ELSE ´R :== CALL Fac(´N − 1);;
´R :== ´N ∗ ´R
FI

Now let us prove that our implementation of *Fac* meets its specification.

lemma (**in** *Fac-impl*)
shows $\forall n.\ \Gamma\vdash \{\!\!|\ ´N = n\ |\!\!\}\ ´R :== PROC\ Fac(´N)\ \{\!\!|\ ´R = fac\ n\ |\!\!\}$
apply (*hoare-rule HoarePartial.ProcRec1*)

1. $\forall n.\ \Gamma,(\bigcup_n \{(\{\!\!|\ ´N = n\ |\!\!\},\ Fac\text{-}'proc,\ \{\!\!|\ ´R = fac\ n\ |\!\!\},\ \emptyset)\})$
$\vdash \{\!\!|\ ´N = n\ |\!\!\}$
$IF\ ´N = 0\ THEN\ ´R :== 1$
$ELSE\ ´R :== CALL\ Fac(´N - 1);;\ ´R :== ´N * ´R\ FI$
$\{\!\!|\ ´R = fac\ n\ |\!\!\}$

apply *vcg*

1. $\bigwedge N.\ (N = 0 \longrightarrow 1 = fac\ N) \wedge$
$(N \neq 0 \longrightarrow (\forall R.\ R = fac\ (N - 1) \longrightarrow N * fac\ (N - 1) = fac\ N))$

apply *simp*
done

Since the factorial is implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from the general rule for mutually recursive procedures):

$$[\![\forall Z.\ \Gamma,\Theta \cup (\bigcup_Z \{(P\ Z,\ p,\ Q\ Z,\ A\ Z)\})\vdash_{/F} (P\ Z)\ \mathit{the}\ (\Gamma\ p)\ (Q\ Z),(A\ Z);\ p \in \mathit{dom}\ \Gamma]\!] \Longrightarrow \forall Z.\ \Gamma,\Theta\vdash_{/F} (P\ Z)\ \mathit{Call}\ p\ (Q\ Z),(A\ Z)$$

The verification condition generator infers the specification out of the context Θ when it encounters a recursive call of the factorial.

26.3 Global Variables and Heap

Now we define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

hoarestate *globals-heap* =
  next :: *ref* ⇒ *ref*
  cont :: *ref* ⇒ *nat*

It is mandatory to start the state name with 'globals'. This is exploited by the syntax translations to store the components in the *globals* part of the state.

Updates to global components inside a procedure are always propagated to the caller. This is implicitly done by the parameter passing syntax translations.

We first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter. The statespace of the global variables has to be imported.

procedures (**imports** *globals-heap*)
  append(*p* :: *ref*, *q*::*ref* | *p*::*ref*)
    IF ´p=Null THEN ´p :== ´q
     ELSE ´p→´next :== CALL append(´p→´next,´q) FI

The difference of a global and a local variable is that global variables are automatically copied back to the procedure caller. We can study this effect on the translation of *´p :== CALL append(´p,´q)*:

call
 (*λs. s(|locals := locals s⟨p-'append-' := locals s·p-'append-',*
   q-'append-' := locals s·q-'append-'⟩|))
 append-'proc (*λs t. s(|globals := globals t|)*)
 (*λi t. ´p :== locals t·p-'append-'*)

Below we give two specifications this time. One captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the second one is a pure frame condition.

The functional specification below introduces two logical variables besides the state space variable σ, namely *Ps* and *Qs*. They are universally quantified and range over both the pre-and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax $\{\!|\sigma.\ \ldots|\!\}$ is a shorthand to fix the current state: $\{s.\ \sigma = s\ \ldots\}$. Moreover ${}^{\sigma}x$ abbreviates the lookup of variable x in the state σ.

The approach to specify procedures on lists basically follows [5]. From the pointer structure in the heap we (relationally) abstract to HOL lists of references. Then we can specify further properties on the level of HOL lists, rather then on the heap. The basic abstractions are:

$Path\ x\ h\ y\ []\ = (x = y)$
$Path\ x\ h\ y\ (p \cdot ps) = (x = p \wedge x \neq Null \wedge Path\ (h\ x)\ h\ y\ ps)$

$Path\ (x{::}ref)\ (h{::}ref \Rightarrow ref)\ (y{::}ref)\ (ps{::}ref\ list)$: *ps* is a list of references that we can obtain out of the heap h by starting with the reference x, following the references in h up to the reference y.

$List\ p\ h\ ps = Path\ p\ h\ Null\ ps$

A list *List p h ps* is a path starting in p and ending up in *Null*.

lemma (**in** *append-impl*) *append-spec1*:
shows $\forall \sigma\ Ps\ Qs.$
  $\Gamma \vdash \{\!|\sigma.\ List\ {}'p\ {}'next\ Ps \wedge\ List\ {}'q\ {}'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}|\!\}$
    $'p :== PROC\ append({}'p,{}'q)$
   $\{\!|List\ {}'p\ {}'next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow {}'next\ x = {}^{\sigma}next\ x)|\!\}$
apply (*hoare-rule HoarePartial.ProcRec1*)

1. $\forall \sigma\ Ps\ Qs.$
   $\Gamma,(\bigcup_{\sigma\ Ps\ Qs}$
     $\{(\{\!|\sigma.\ List\ {}'p\ {}'next\ Ps \wedge List\ {}'q\ {}'next\ Qs \wedge set\ Ps \cap set\ Qs = \emptyset|\!\},$
      $append\text{-}'proc,$
      $\{\!|List\ {}'p\ {}'next\ (Ps\ @\ Qs) \wedge$
       $(\forall x.\ x \notin set\ Ps \longrightarrow x \rightarrow {}'next = {}^{\sigma}next\ x)|\!\},$
      $\emptyset)\})$
    $\vdash \{\!|\sigma.\ List\ {}'p\ {}'next\ Ps \wedge List\ {}'q\ {}'next\ Qs \wedge set\ Ps \cap set\ Qs = \emptyset|\!\}$
     $IF\ {}'p = Null\ THEN\ {}'p :== {}'q$
     $ELSE\ {}'p \rightarrow {}'next :== CALL\ append({}'p \rightarrow {}'next,{}'q)\ FI$
     $\{\!|List\ {}'p\ {}'next\ (Ps\ @\ Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow x \rightarrow {}'next = {}^{\sigma}next\ x)|\!\}$

Note that *hoare-rule* takes care of multiple auxiliary variables! *HoarePartial.ProcRec1* has only one auxiliary variable, namely Z. But the type of Z can be instantiated arbitrarily. So *hoare-rule* instantiates Z with the tuple $(\sigma,\ Ps,\ Qs)$ and derives a proper variant of the rule. Therefore *hoare-rule* depends on the proper quantification of auxiliary variables!

apply *vcg*

$$
\begin{array}{l}
1.\ \bigwedge Ps\ Qs\ next\ p\ q.\\
\quad [\![List\ p\ next\ Ps;\ List\ q\ next\ Qs;\ set\ Ps \cap set\ Qs = \emptyset]\!]\\
\quad \Longrightarrow (p = Null \longrightarrow\\
\qquad\quad List\ q\ next\ (Ps\ @\ Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow next\ x = next\ x)) \wedge\\
\qquad (p \neq Null \longrightarrow\\
\qquad\ (\exists Psa.\ List\ (next\ p)\ next\ Psa \wedge\\
\qquad\qquad (\exists Qsa.\ List\ q\ next\ Qsa \wedge\\
\qquad\qquad\qquad set\ Psa \cap set\ Qsa = \emptyset \wedge\\
\qquad\qquad\qquad (\forall nexta\ pa.\\
\qquad\qquad\qquad\quad List\ pa\ nexta\ (Psa\ @\ Qsa) \wedge\\
\qquad\qquad\qquad\quad (\forall x.\ x \notin set\ Psa \longrightarrow nexta\ x = next\ x) \longrightarrow\\
\qquad\qquad\qquad\quad List\ p\ (nexta(p := pa))\ (Ps\ @\ Qs) \wedge\\
\qquad\qquad\qquad\quad (\forall x.\ x \notin set\ Ps \longrightarrow\\
\qquad\qquad\qquad\qquad (nexta(p := pa))\ x = next\ x)))))
\end{array}
$$

For each branch of the *IF* statement we have one conjunct to prove. The *THEN* branch starts with $p = Null \longrightarrow \ldots$ and the *ELSE* branch with $p \neq Null \longrightarrow \ldots$. Let us focus on the *ELSE* branch, were the recursive call to append occurs. First of all we have to prove that the precondition for the recursive call is fulfilled. That means we have to provide some witnesses for the lists *Psa* and *Qsa* which are referenced by $p \rightarrow next$ (now written as *next p*) and *q*. Then we have to show that we can derive the overall postcondition from the postcondition of the recursive call. The state components that have changed by the recursive call are the ones with the suffix *a*, like *nexta* and *pa*.

apply *fastforce*
done

If the verification condition generator works on a procedure call it checks whether it can find a modifies clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoarePartial.ProcSpec* is applied. Simplification of the procedure call means that the "copy back" of the global components is simplified. Only those components that occur in the modifies clause are actually copied back. This simplification is justified by the rule *HoarePartial.ProcModifyReturn*. So after this simplification all global components that do not appear in the modifies clause are treated as local variables.

We study the effect of the modifies clause on the following examples, where we want to prove that (@) does not change the *cont* part of the heap.

lemma (**in** *append-impl*)
shows $\Gamma\vdash \{\!|\ \acute{}cont{=}c|\!\}\ \acute{}p :== CALL\ append(Null,Null)\ \{\!|\ \acute{}cont{=}c|\!\}$
proof –

note *append-spec* = *append-spec1*
show *?thesis*
apply *vcg*

$$
\begin{array}{l}
1.\ \bigwedge cont\ next.\\
\quad \exists\, Ps.\ List\ Null\ next\ Ps\ \wedge\\
\qquad (\exists\, Qs.\ List\ Null\ next\ Qs\ \wedge\\
\qquad\quad set\ Ps \cap set\ Qs = \emptyset\ \wedge\\
\qquad\quad (\forall\, conta\ nexta\ p.\\
\qquad\qquad List\ p\ nexta\ (Ps\ @\ Qs)\ \wedge\\
\qquad\qquad (\forall\, x.\ x \notin set\ Ps \longrightarrow nexta\ x = next\ x) \longrightarrow\\
\qquad\qquad conta = cont))
\end{array}
$$

Only focus on the very last line: *conta* is the heap component after the procedure call, and *cont* the heap component before the procedure call. Since we have not added the modified clause we do not know that they have to be equal.

oops

We now add the frame condition. The list in the modifies clause names all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts are not changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state t here.

lemma (**in** *append-impl*) *append-modifies*:
shows $\forall\sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}$ *´p :== PROC append(´p,´q)*
$\{t.\ t$ *may-only-modify-globals* σ *in* $[next]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

We tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies*. It also tries to solve the verification conditions automatically. Again it is crucial to name the lemma with this naming scheme, since the verfication condition generator searches for these names.

The modifies clause is equal to a state update specification of the following form.

lemma (**in** *append-impl*) **shows** $\{t.\ t$ *may-only-modify-globals* Z *in* $[next]\}$
=
$\{t.\ \exists\, next.\ globals\ t{=}update\ id\ id\ next\text{-}'\ (K\text{-}statefun\ next)\ (globals\ Z)\}$
apply (*unfold mex-def meq-def*)
apply *simp*
done

Now that we have proven the frame-condition, it is available within the locale *append-impl* and the *vcg* exploits it.

lemma (**in** *append-impl*)
shows $\Gamma\vdash \{\!|\,\acute{}cont{=}c|\!\}\ \acute{}p :== CALL\ append(Null,Null)\ \{\!|\,\acute{}cont{=}c|\!\}$
proof −
  note *append-spec* = *append-spec1*
  show *?thesis*
    apply *vcg*

$$
\begin{array}{l}
1.\ \bigwedge cont\ next.\\
\quad\quad \exists Ps.\ List\ Null\ next\ Ps\ \wedge\\
\quad\quad\quad\quad (\exists Qs.\ List\ Null\ next\ Qs\ \wedge\\
\quad\quad\quad\quad\quad\quad set\ Ps \cap set\ Qs = \emptyset\ \wedge\\
\quad\quad\quad\quad\quad\quad (\forall nexta\ p.\\
\quad\quad\quad\quad\quad\quad\quad List\ p\ nexta\ (Ps\ @\ Qs)\ \wedge\\
\quad\quad\quad\quad\quad\quad\quad (\forall x.\ x \notin set\ Ps \longrightarrow nexta\ x = next\ x) \longrightarrow\\
\quad\quad\quad\quad\quad\quad\quad cont = cont))
\end{array}
$$

With a modifies clause present we know that no change to *cont* has occurred.

    by *simp*
qed

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but we may need the modifies clause to prove the functional specifications. So usually the modifies clause is proved before the proof of the functional specification, so that it can already be used by the verification condition generator.

26.4 Total Correctness

When proving total correctness the additional proof burden to the user is to come up with a well-founded relation and to prove that certain states get smaller according to this relation. Proving that a relation is well-founded can be quite hard. But fortunately there are ways to construct and stick together relations so that they are well-founded by construction. This infrastructure is already present in Isabelle/HOL. For example, *measure f* is always well-founded; the lexicographic product of two well-founded relations is again well-founded and the inverse image construction *inv-image* of a well-founded relation is again well-founded. The constructions are best explained by some equations:

$((x, y) \in \textit{measure } f) = (f\, x < f\, y)$
$(((a, b), x, y) \in r \mathbin{<\!*\textit{lex}*\!>} s) = ((a, x) \in r \lor a = x \land (b, y) \in s)$
$((x, y) \in \textit{inv-image } r\, f) = ((f\, x, f\, y) \in r)$

Another useful construction is $<\!*\textit{mlex}*\!>$ which is a combination of a measure and a lexicographic product:

$((x, y) \in f \mathbin{<\!*\textit{mlex}*\!>} r) = (f\, x < f\, y \lor f\, x = f\, y \land (x, y) \in r)$

In contrast to the lexicographic product it does not construct a product type. The state may either decrease according to the measure function f or the measure stays the same and the state decreases because of the relation r.

Lets look at a loop:

lemma (**in** *vars*)
  $\Gamma \vdash_t \{\!|\,\acute{}M = 0 \land \acute{}S = 0\,|\!\}$
    $\textit{WHILE } \acute{}M \neq a$
    $\textit{INV } \{\!|\,\acute{}S = \acute{}M * b \land \acute{}M \leq a\,|\!\}$
    $\textit{VAR MEASURE } a - \acute{}M$
    $\textit{DO } \acute{}S :== \acute{}S + b;; \acute{}M :== \acute{}M + 1 \textit{ OD}$
    $\{\!|\,\acute{}S = a * b\,|\!\}$
apply *vcg*

1. $\bigwedge M\, S.\ [\![M = 0; S = 0]\!] \Longrightarrow S = M * b \land M \leq a$
2. $\bigwedge M\, S.\ [\![S = M * b; M \leq a; M \neq a]\!]$
   $\Longrightarrow a - (M + 1) < a - M \land S + b = (M + 1) * b \land M + 1 \leq a$
3. $\bigwedge M\, S.\ [\![S = M * b; M \leq a; \neg\, M \neq a]\!] \Longrightarrow S = a * b$

The first conjunct of the second subgoal is the proof obligation that the variant decreases in the loop body.

by *auto*

The variant annotation is preceded by *VAR*. The capital *MEASURE* is a shorthand for $\textit{measure } (\lambda s.\ a - {}^{s}M)$. Analogous there is a capital $<\!*\textit{MLEX}*\!>$.

lemma (**in** *Fac-impl*) *Fac-spec′*:
shows $\forall \sigma.\ \Gamma \vdash_t \{\sigma\}\ \acute{}R :== \ \textit{PROC Fac}(\acute{}N)\ \{\!|\,\acute{}R = \textit{fac } {}^{\sigma}N\,|\!\}$
apply (*hoare-rule HoareTotal.ProcRec1* [**where** r=*measure* $(\lambda(s,p).\ {}^{s}N)$])

In case of the factorial the parameter N decreases in every call. This is easily expressed by the measure function. Note that the well-founded relation for recursive procedures is formally defined on tuples containing the state space and the procedure name.

1. $\forall \sigma\ \sigma'.$
   $\Gamma,(\bigcup_{\sigma'} \{(\{\sigma'\} \cap \{\!|\,\acute{}N < {}^{\sigma}N\,|\!\}, \textit{Fac-}'\textit{proc}, \{\!|\,\acute{}R = \textit{fac } {}^{\sigma'}N\,|\!\}, \emptyset)\})$
     $\vdash_t (\{\sigma\} \cap \{\sigma'\})$

$IF\ ´N = 0\ THEN\ ´R :== 1$
$ELSE\ ´R :== CALL\ Fac(´N - 1);;\ ´R :== ´N * ´R\ FI$
$\{\!|´R = fac\ {}^{\sigma'}N|\!\}$

The initial call to the factorial is in state σ. Note that in the precondition $\{\sigma\} \cap \{\sigma'\}$, σ' stems from the lemma we want to prove and σ stems from the recursion rule for total correctness. Both are synonym for the initial state. To use the assumption in the Hoare context we have to show that the call to the factorial is invoked on a smaller N compared to the initial ${}^{\sigma}N$.

apply *vcg*

$$1.\ \bigwedge N.\ (N = 0 \longrightarrow 1 = fac\ N) \wedge$$
$$(N \neq 0 \longrightarrow$$
$$N - 1 < N \wedge (\forall R.\ R = fac\ (N - 1) \longrightarrow N * fac\ (N - 1) = fac\ N))$$

The tribute to termination is that we have to show $N - 1 < N$ in case of the recursive call.

by *simp*

lemma (**in** *append-impl*) *append-spec2*:
shows $\forall \sigma\ Ps\ Qs.\ \Gamma\vdash_t$
$\{\!|\sigma.\ List\ ´p\ ´next\ Ps \wedge\ List\ ´q\ ´next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}|\!\}$
$´p :== PROC\ append(´p,´q)$
$\{\!|List\ ´p\ ´next\ (Ps@Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow ´next\ x = {}^{\sigma}next\ x)|\!\}$
apply (*hoare-rule HoareTotal.ProcRec1*
[**where** r=*measure* $(\lambda(s,p).\ length\ (list\ {}^{s}p\ {}^{s}next))$])

In case of the append function the length of the list referenced by p decreases in every recursive call.

$1.\ \forall \sigma\ \sigma'\ Ps\ Qs.$
$\Gamma,(\bigcup_{\sigma'\ Ps\ Qs}$
$\{(\{\!|\sigma'.\ List\ ´p\ ´next\ Ps \wedge List\ ´q\ ´next\ Qs \wedge set\ Ps \cap set\ Qs = \emptyset|\!\} \cap$
$\{\!|\,|list\ ´p\ ´next| < |list\ {}^{\sigma}p\ {}^{\sigma}next|\,|\!\},$
$append\text{-}'proc,$
$\{\!|List\ ´p\ ´next\ (Ps\ @\ Qs) \wedge$
$(\forall x.\ x \notin set\ Ps \longrightarrow x\rightarrow´next = {}^{\sigma'}next\ x)|\!\},$
$\emptyset)\})$
$\vdash_t (\{\sigma\} \cap$
$\{\!|\sigma'.\ List\ ´p\ ´next\ Ps \wedge List\ ´q\ ´next\ Qs \wedge set\ Ps \cap set\ Qs = \emptyset|\!\})$
$IF\ ´p = Null\ THEN\ ´p :== ´q$
$ELSE\ ´p\rightarrow´next :== CALL\ append(´p\rightarrow´next,´q)\ FI$
$\{\!|List\ ´p\ ´next\ (Ps\ @\ Qs) \wedge (\forall x.\ x \notin set\ Ps \longrightarrow x\rightarrow´next = {}^{\sigma'}next\ x)|\!\}$

apply *vcg*
apply (*fastforce simp add*: *List-list*)

done

In case of the lists above, we have used a relational list abstraction *List* to construct the HOL lists *Ps* and *Qs* for the pre- and postcondition. To supply a proper measure function we use a functional abstraction *list*. The functional abstraction can be defined by means of the relational list abstraction, since the lists are already uniquely determined by the relational abstraction:

$islist\ p\ h = (\exists\, ps.\ List\ p\ h\ ps)$
$list\ p\ h = (THE\ ps.\ List\ p\ h\ ps)$
lemma $List\ p\ h\ ps = (islist\ p\ h \wedge ps = list\ p\ h)$

The next contrived example is taken from [3], to illustrate a more complex termination criterion for mutually recursive procedures. The procedures do not calculate anything useful.

procedures
  pedal(*N*::*nat*,*M*::*nat*)
  IF 0 < ´N THEN
    IF 0 < ´M THEN
      CALL coast(*´N− 1*,*´M− 1*) *FI*;;
      CALL pedal(*´N− 1*,*´M*)
    FI
  and

  coast(*N*::*nat*,*M*::*nat*)
  CALL pedal(*´N*,*´M*);;
   IF 0 < ´M THEN CALL coast(*´N*,*´M− 1*) *FI*

In the recursive calls in procedure *pedal* the first argument always decreases. In the body of *coast* in the recursive call of *coast* the second argument decreases, but in the call to *pedal* no argument decreases. Therefore an relation only on the state space is insufficient. We have to take the procedure names into account, too. We consider the procedure *coast* to be "bigger" than *pedal* when we construct a well-founded relation on the product of state space and procedure names.

ML ‹*ML-Thms.bind-thm* (*HoareTotal-ProcRec2*, *Hoare.gen-proc-rec* @{*context*} *Hoare.Total 2*)›

We provide the ML function `gen_proc_rec` to automatically derive a convenient rule for recursion for a given number of mutually recursive procedures.

lemma (**in** *pedal-coast-clique*)
shows $(\forall \sigma.\ \Gamma\vdash_t \{\sigma\}$ *PROC pedal*(*´N*,*´M*) *UNIV*) $\wedge$
        $(\forall \sigma.\ \Gamma\vdash_t \{\sigma\}$ *PROC coast*(*´N*,*´M*) *UNIV*)
apply (*hoare-rule HoareTotal-ProcRec2*
        [**where** *r*= $((\lambda(s,p).\ {}^sN)$ *<∗mlex∗>*
                $(\lambda(s,p).\ {}^sM)$ *<∗mlex∗>*
                measure $(\lambda(s,p).$ *if p = coast-'proc then 1 else 0*$))])$

We can directly express the termination condition described above with the <∗*mlex*∗> construction. Either state component N decreases, or it stays the same and M decreases or this also stays the same, but then the procedure name has to decrease.

$$
\begin{array}{l}
\mathit{1.}\ \forall \sigma\ \sigma'.\ \Gamma,(\bigcup_{\sigma'}\ \{(\{\sigma'\} \cap \\
\qquad\qquad \{\!|\ \acute{}N < {}^{\sigma}N \lor \\
\qquad\qquad\quad \acute{}N = {}^{\sigma}N \land \\
\qquad\qquad\quad (\acute{}M < {}^{\sigma}M \lor \\
\qquad\qquad\quad\ \acute{}M = {}^{\sigma}M \land \\
\qquad\qquad\quad\ (\mathsf{if}\ \mathit{pedal\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}) \\
\qquad\qquad\quad\ < (\mathsf{if}\ \mathit{pedal\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}))|\!\}, \\
\qquad\qquad \mathit{pedal\text{-}'proc},\ \mathit{UNIV},\ \emptyset)\}) \cup \\
\qquad (\bigcup_{\sigma'}\ \{(\{\sigma'\} \cap \\
\qquad\qquad \{\!|\ \acute{}N < {}^{\sigma}N \lor \\
\qquad\qquad\quad \acute{}N = {}^{\sigma}N \land \\
\qquad\qquad\quad (\acute{}M < {}^{\sigma}M \lor \\
\qquad\qquad\quad\ \acute{}M = {}^{\sigma}M \land \\
\qquad\qquad\quad\ (\mathsf{if}\ \mathit{coast\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}) \\
\qquad\qquad\quad\ < (\mathsf{if}\ \mathit{pedal\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}))|\!\}, \\
\qquad\qquad \mathit{coast\text{-}'proc},\ \mathit{UNIV},\ \emptyset)\}) \\
\qquad \vdash_t (\{\sigma\} \cap \{\sigma'\}) \\
\qquad\quad \mathit{IF}\ \mathit{0} < \acute{}N \\
\qquad\quad \mathit{THEN}\ \mathit{IF}\ \mathit{0} < \acute{}M\ \mathit{THEN}\ \mathit{CALL}\ \mathit{coast}(\acute{}N - \mathit{1}, \acute{}M - \mathit{1})\ \mathit{FI};; \\
\qquad\quad\ \ \mathit{CALL}\ \mathit{pedal}(\acute{}N - \mathit{1}, \acute{}M) \\
\qquad\quad \mathit{FI} \\
\qquad\quad \mathit{UNIV} \\
\mathit{2.}\ \forall \sigma\ \sigma'.\ \Gamma,(\bigcup_{\sigma'}\ \{(\{\sigma'\} \cap \\
\qquad\qquad \{\!|\ \acute{}N < {}^{\sigma}N \lor \\
\qquad\qquad\quad \acute{}N = {}^{\sigma}N \land \\
\qquad\qquad\quad (\acute{}M < {}^{\sigma}M \lor \\
\qquad\qquad\quad\ \acute{}M = {}^{\sigma}M \land \\
\qquad\qquad\quad\ (\mathsf{if}\ \mathit{pedal\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}) \\
\qquad\qquad\quad\ < (\mathsf{if}\ \mathit{coast\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}))|\!\}, \\
\qquad\qquad \mathit{pedal\text{-}'proc},\ \mathit{UNIV},\ \emptyset)\}) \cup \\
\qquad (\bigcup_{\sigma'}\ \{(\{\sigma'\} \cap \\
\qquad\qquad \{\!|\ \acute{}N < {}^{\sigma}N \lor \\
\qquad\qquad\quad \acute{}N = {}^{\sigma}N \land \\
\qquad\qquad\quad (\acute{}M < {}^{\sigma}M \lor \\
\qquad\qquad\quad\ \acute{}M = {}^{\sigma}M \land \\
\qquad\qquad\quad\ (\mathsf{if}\ \mathit{coast\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}) \\
\qquad\qquad\quad\ < (\mathsf{if}\ \mathit{coast\text{-}'proc} = \mathit{coast\text{-}'proc}\ \mathsf{then}\ \mathit{1}\ \mathsf{else}\ \mathit{0}))|\!\}, \\
\qquad\qquad \mathit{coast\text{-}'proc},\ \mathit{UNIV},\ \emptyset)\}) \\
\qquad \vdash_t (\{\sigma\} \cap \{\sigma'\}) \\
\qquad\quad \mathit{CALL}\ \mathit{pedal}(\acute{}N, \acute{}M);;\ \mathit{IF}\ \mathit{0} < \acute{}M\ \mathit{THEN}\ \mathit{CALL}\ \mathit{coast}(\acute{}N, \acute{}M - \mathit{1}) \\
\mathit{FI} \\
\qquad\quad \mathit{UNIV}
\end{array}
$$

apply *simp-all*

1. $\forall \sigma\ \sigma'.$
Γ,($\bigcup_x$ {({x} $\cap$ $\{\!|$ $\acute{}N$ $<$ $^{\sigma}N$ $\vee$ $\acute{}N$ $=$ $^{\sigma}N$ $\wedge$ $\acute{}M$ $<$ $^{\sigma}M|\!\}$, *pedal-'proc*, *UNIV*, $\emptyset$)}) $\cup$
($\bigcup_x$ {({x} $\cap$ $\{\!|$ $\acute{}N$ $<$ $^{\sigma}N$ $\vee$ $\acute{}N$ $=$ $^{\sigma}N$ $\wedge$ $\acute{}M$ $<$ $^{\sigma}M|\!\}$, *coast-'proc*, *UNIV*, $\emptyset$)})
$\vdash_t$ ({σ} $\cap$ {σ'})
IF 0 $<$ $\acute{}N$
THEN IF 0 $<$ $\acute{}M$ *THEN CALL coast*($\acute{}N$ $-$ *Suc 0*,$\acute{}M$ $-$ *Suc 0*) *FI*;;
CALL pedal($\acute{}N$ $-$ *Suc 0*,$\acute{}M$)
FI
UNIV

2. $\forall \sigma\ \sigma'.$
Γ,($\bigcup_x$ {({x} $\cap$ $\{\!|$ $\acute{}N$ $<$ $^{\sigma}N$ $\vee$ $\acute{}N$ $=$ $^{\sigma}N$ $\wedge$ ($\acute{}M$ $<$ $^{\sigma}M$ $\vee$ $\acute{}M$ $=$ $^{\sigma}M$)$|\!\}$, *pedal-'proc*, *UNIV*, $\emptyset$)}) $\cup$
($\bigcup_x$ {({x} $\cap$ $\{\!|$ $\acute{}N$ $<$ $^{\sigma}N$ $\vee$ $\acute{}N$ $=$ $^{\sigma}N$ $\wedge$ $\acute{}M$ $<$ $^{\sigma}M|\!\}$, *coast-'proc*, *UNIV*, $\emptyset$)})
$\vdash_t$ ({σ} $\cap$ {σ'})
CALL pedal($\acute{}N$,$\acute{}M$);;
IF 0 $<$ $\acute{}M$ *THEN CALL coast*($\acute{}N$,$\acute{}M$ $-$ *Suc 0*) *FI*
UNIV

by (*vcg*,*simp*)+

We can achieve the same effect without <$*$*mlex*$*$> by using the ordinary lexicographic product <$*$*lex*$*$>, *inv-image* and *measure*

lemma (**in** *pedal-coast-clique*)
shows ($\forall \sigma$. $\Gamma\vdash_t$ {σ} *PROC pedal*($\acute{}N$,$\acute{}M$) *UNIV*) $\wedge$
($\forall \sigma$. $\Gamma\vdash_t$ {σ} *PROC coast*($\acute{}N$,$\acute{}M$) *UNIV*)
apply (*hoare-rule HoareTotal-ProcRec2*
[**where** r= *inv-image* (*measure* ($\lambda m.$ m) <$*$*lex*$*$>
measure ($\lambda m.$ m) <$*$*lex*$*$>
measure ($\lambda p.$ *if* p = *coast-'proc then 1 else 0*))
($\lambda(s,p)$. (^{s}N,^{s}M,p))])

With the lexicographic product we construct a well-founded relation on triples of type *nat* $\times$ *nat* $\times$ *string*. With *inv-image* we project the components out of the state-space and the procedure names to this triple.

1. $\forall \sigma\ \sigma'.$
Γ,($\bigcup_{\sigma'}$ {({σ'} $\cap$
$\{\!|$ $\acute{}N$ $<$ $^{\sigma}N$ $\vee$
$\acute{}N$ $=$ $^{\sigma}N$ $\wedge$
($\acute{}M$ $<$ $^{\sigma}M$ $\vee$
$\acute{}M$ $=$ $^{\sigma}M$ $\wedge$
(*if pedal-'proc* $=$ *coast-'proc then 1 else 0*)
$<$ (*if pedal-'proc* $=$ *coast-'proc then 1 else 0*))$|\!\}$,
pedal-'proc, *UNIV*, $\emptyset$)}) $\cup$
($\bigcup_{\sigma'}$ {({σ'} $\cap$

$\{\!|\, ´N < {}^{\sigma}N \vee$
$´N = {}^{\sigma}N \wedge$
$(´M < {}^{\sigma}M \vee$
$´M = {}^{\sigma}M \wedge$
$(\textsf{if}\ coast\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0)$
$< (\textsf{if}\ pedal\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0))|\!\},$
$coast\text{-}'proc,\ UNIV,\ \emptyset)\})$
$\vdash_t (\{\sigma\} \cap \{\sigma'\})$
$IF\ 0 < ´N$
$THEN\ IF\ 0 < ´M\ THEN\ CALL\ coast(´N - 1, ´M - 1)\ FI;;$
$CALL\ pedal(´N - 1, ´M)$
FI
$UNIV$

2. $\forall \sigma\ \sigma'.$
$\Gamma, (\bigcup_{\sigma'} \{(\{\sigma'\} \cap$
$\{\!|\, ´N < {}^{\sigma}N \vee$
$´N = {}^{\sigma}N \wedge$
$(´M < {}^{\sigma}M \vee$
$´M = {}^{\sigma}M \wedge$
$(\textsf{if}\ pedal\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0)$
$< (\textsf{if}\ coast\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0))|\!\},$
$pedal\text{-}'proc,\ UNIV,\ \emptyset)\}) \cup$
$(\bigcup_{\sigma'} \{(\{\sigma'\} \cap$
$\{\!|\, ´N < {}^{\sigma}N \vee$
$´N = {}^{\sigma}N \wedge$
$(´M < {}^{\sigma}M \vee$
$´M = {}^{\sigma}M \wedge$
$(\textsf{if}\ coast\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0)$
$< (\textsf{if}\ coast\text{-}'proc = coast\text{-}'proc\ \textsf{then}\ 1\ \textsf{else}\ 0))|\!\},$
$coast\text{-}'proc,\ UNIV,\ \emptyset)\})$
$\vdash_t (\{\sigma\} \cap \{\sigma'\})$
$CALL\ pedal(´N, ´M);;\ IF\ 0 < ´M\ THEN\ CALL\ coast(´N, ´M - 1)\ FI$
$UNIV$

apply *simp-all*
by (*vcg*,*simp*)+

By doing some arithmetic we can express the termination condition with a single measure function.

lemma (**in** *pedal-coast-clique*)
shows $(\forall \sigma.\ \Gamma \vdash_t \{\sigma\}\ PROC\ pedal(´N, ´M)\ UNIV) \wedge$
$(\forall \sigma.\ \Gamma \vdash_t \{\sigma\}\ PROC\ coast(´N, ´M)\ UNIV)$
apply(*hoare-rule HoareTotal-ProcRec2*
[**where** $r = measure\ (\lambda(s,p).\ {}^{s}N + {}^{s}M + (if\ p = coast\text{-}'proc\ then\ 1\ else\ 0))$])
apply *simp-all*

1. $\forall \sigma\ \sigma'.$
$\Gamma, (\bigcup_x \{(\{x\} \cap \{\!|\, ´N + ´M < {}^{\sigma}N + {}^{\sigma}M|\!\},\ pedal\text{-}'proc,\ UNIV,\ \emptyset)\}) \cup$

$(\bigcup_x \{(\{x\} \cap \{\!|Suc\ (\acute{}N + \acute{}M) < {}^{\sigma}N + {}^{\sigma}M|\!\},\ coast\text{-}'proc,\ UNIV,\ \emptyset)\})$
$\vdash_t (\{\sigma\} \cap \{\sigma'\})$
IF 0 < ´N
THEN IF 0 < ´M THEN CALL coast(´N − Suc 0,´M − Suc 0) FI;;
CALL pedal(´N − Suc 0,´M)
FI
UNIV

2. $\forall \sigma\ \sigma'.$
$\Gamma,(\bigcup_x \{(\{x\} \cap \{\!|\acute{}N + \acute{}M < Suc\ ({}^{\sigma}N + {}^{\sigma}M)|\!\},\ pedal\text{-}'proc,\ UNIV,\ \emptyset)\}) \cup$
$(\bigcup_x \{(\{x\} \cap \{\!|\acute{}N + \acute{}M < {}^{\sigma}N + {}^{\sigma}M|\!\},\ coast\text{-}'proc,\ UNIV,\ \emptyset)\})$
$\vdash_t (\{\sigma\} \cap \{\sigma'\})$
CALL pedal(´N,´M);;
IF 0 < ´M THEN CALL coast(´N,´M − Suc 0) FI
UNIV

by (*vcg*,*simp*,*arith?*)+

26.5 Guards

The purpose of a guard is to guard the **(sub-) expressions** of a statement against runtime faults. Typical runtime faults are array bound violations, dereferencing null pointers or arithmetical overflow. Guards make the potential runtime faults explicit, since the expressions themselves never "fail" because they are ordinary HOL expressions. To relieve the user from typing in lots of standard guards for every subexpression, we supply some input syntax for the common language constructs that automatically generate the guards. For example the guarded assignment $\acute{}M :==_g (\acute{}M + 1)\ div\ \acute{}N$ gets expanded to guarded command $(False,\ \{\!|in\text{-}range\ (\acute{}M + 1) \wedge \acute{}N \neq 0 \wedge in\text{-}range\ ((\acute{}M + 1)\ div\ \acute{}N)|\!\}) \longmapsto \acute{}M :== (\acute{}M + 1)\ div\ \acute{}N$. Here *in-range* is uninterpreted by now.

lemma (**in** *vars*) $\Gamma\vdash\{\!|True|\!\}\ \acute{}M :==_g (\acute{}M + 1)\ div\ \acute{}N\ \{\!|True|\!\}$
apply *vcg*

1. $\bigwedge M\ N.\ True \Longrightarrow in\text{-}range\ (M + 1) \wedge N \neq 0 \wedge in\text{-}range\ ((M + 1)\ div\ N)$

oops

The user can supply on (overloaded) definition of *in-range* to fit to his needs. Currently guards are generated for:

- overflow and underflow of numbers (*in-range*). For subtraction of natural numbers $a - b$ the guard $b \leq a$ is generated instead of *in-range* to guard against underflows.
- division by *0*
- dereferencing of *Null* pointers

- array bound violations

Following (input) variants of guarded statements are available:

- Assignment: $\ldots :==_g \ldots$
- If: $IF_g \ldots$
- While: $WHILE_g \ldots$
- Call: $CALL_g \ldots$ or $\ldots :== CALL_g \ldots$

26.6 Miscellaneous Techniques

26.6.1 Modifies Clause

We look at some issues regarding the modifies clause with the example of insertion sort for heap lists.

primrec *sorted*:: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$
where
sorted le [] = *True* |
sorted le $(x\#xs) = ((\forall y \in set\ xs.\ le\ x\ y) \wedge sorted\ le\ xs)$

procedures (**imports** *globals-heap*)
  insert(*r*::*ref*,*p*::*ref* | *p*::*ref*)
    IF ´r=Null THEN SKIP
     ELSE IF ´p=Null THEN ´p :== ´r;; ´p→´next :== Null
          ELSE IF ´r→´cont ≤ ´p→´cont
               THEN ´r→´next :== ´p;; ´p:==´r
               ELSE ´p→´next :== CALL insert(´r,´p→´next)
               FI
          FI
     FI

lemma (**in** *insert-impl*) *insert-modifies*:
   $\forall \sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}$ *´p :== PROC insert(´r,´p)*
      $\{t.\ t$ *may-only-modify-globals* σ *in* [*next*]$\}$
  by (*hoare-rule HoarePartial.ProcRec1*) (*vcg spec=modifies*)

lemma (**in** *insert-impl*) *insert-spec*:
  $\forall \sigma\ Ps\ .\ \Gamma\vdash$
  $\{\!|\sigma.$ *List ´p ´next Ps* $\wedge$ *sorted* $(\leq)$ (*map ´cont Ps*) $\wedge$
     ´r $\neq$ *Null* $\wedge$ *´r* $\notin$ *set Ps*$|\!\}$
   ´p :== PROC insert(´r,´p)
  $\{\!|\exists Qs.$ *List ´p ´next Qs* $\wedge$ *sorted* $(\leq)$ (*map* $^{\sigma}$*cont Qs*) $\wedge$
      set Qs = insert $^{\sigma}$*r* (*set Ps*) $\wedge$
      $(\forall x.\ x \notin$ *set Qs* $\longrightarrow$ *´next x =* $^{\sigma}$*next x*$)|\!\}$

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state. The reason is that we have separated out the information that *cont* is not modified by the procedure, to the modifies clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitly state an equality in the postcondition. The reason is simple. If the postcondition would talk about *´cont* instead of $^{\sigma}$*cont*, we get a new instance of *cont* during verification and the postcondition would only state something about this new instance. But as the verification condition generator uses the modifies clause the caller of *insert* instead still has the old *cont* after the call. Thats the sense of the modifies clause. So the caller and the specification simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

26.6.2 Annotations

Annotations (like loop invariants) are mere syntactic sugar of statements that are used by the *vcg*. Logically a statement with an annotation is equal to the statement without it. Hence annotations can be introduced by the user while building a proof:

$$HoarePartial.annotateI\colon \frac{\Gamma,\Theta\vdash_{/F} P\ anno\ Q,A \qquad c = anno}{\Gamma,\Theta\vdash_{/F} P\ c\ Q,A}$$

When introducing annotations it can easily happen that these mess around with the nesting of sequential composition. Then after stripping the annotations the resulting statement is no longer syntactically identical to original one, only equivalent modulo associativity of sequential composition. The following rule also deals with this case:

$$HoarePartial.annotate\text{-}normI\colon \frac{\begin{array}{c}\Gamma,\Theta\vdash_{/F} P\ anno\ Q,A\\ Language.normalize\ c = Language.normalize\ anno\end{array}}{\Gamma,\Theta\vdash_{/F} P\ c\ Q,A}$$

Loop Annotations

procedures (**imports** *globals-heap*)
 insertSort(*p*::*ref*| *p*::*ref*)
 where *r*::*ref* *q*::*ref* **in**
  ´r:==*Null*;;
  WHILE (*´p* ≠ *Null*) *DO*
   ´q :== *´p*;;
   ´p :== *´p*→*´next*;;
   ´r :== *CALL insert*(*´q*,*´r*)

OD;;
$\acute{}p$:==$\acute{}r$

lemma (**in** *insertSort-impl*) *insertSort-modifies*:
shows
$\forall\sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}\ \acute{}p :== PROC\ insertSort(\acute{}p)$
$\{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \sigma\ in\ [next]\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*vcg spec=modifies*)
done

Insertion sort is not implemented recursively here, but with a loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. Therefore we annotate the loop first, before we run the *vcg*.

lemma (**in** *insertSort-impl*) *insertSort-spec*:
shows $\forall\sigma\ Ps.$
$\Gamma\vdash \{\!|\sigma.\ List\ \acute{}p\ \acute{}next\ Ps\ |\!\}$
$\acute{}p :== PROC\ insertSort(\acute{}p)$
$\{\!|\exists\, Qs.\ List\ \acute{}p\ \acute{}next\ Qs \wedge sorted\ (\le)\ (map\ {}^{\sigma}cont\ Qs) \wedge$
$set\ Qs = set\ Ps|\!\}$
apply (*hoare-rule HoarePartial.ProcRec1*)
apply (*hoare-rule anno=*
$\acute{}r :== Null;;$
$WHILE\ \acute{}p \neq Null$
$INV\ \{\!|\exists\, Qs\ Rs.\ List\ \acute{}p\ \acute{}next\ Qs \wedge List\ \acute{}r\ \acute{}next\ Rs \wedge$
$set\ Qs \cap set\ Rs = \{\} \wedge$
$sorted\ (\le)\ (map\ \acute{}cont\ Rs) \wedge set\ Qs \cup set\ Rs = set\ Ps \wedge$
$\acute{}cont = {}^{\sigma}cont\ |\!\}$
$DO\ \acute{}q :== \acute{}p;;\ \acute{}p :== \acute{}p\to\acute{}next;;\ \acute{}r :== CALL\ insert(\acute{}q,\acute{}r)\ OD;;$
$\acute{}p :== \acute{}r$ **in** *HoarePartial.annotateI*)
apply *vcg*

...

The method *hoare-rule* automatically solves the side-condition that the annotated program is the same as the original one after stripping the annotations.

Specification Annotations

When verifying a larger block of program text, it might be useful to split up the block and to prove the parts in isolation. This is especially useful to isolate loops. On the level of the Hoare calculus the parts can then be combined with the consequence rule. To automate this process we introduce the derived command *specAnno*, which allows to introduce a Hoare tuple (inclusive auxiliary variables) in the program text:

specAnno $P\ c\ Q\ A = c\ undefined$

The whole annotation reduces to the body c *undefined*. The type of the assertions P, Q and A is $'a \Rightarrow 's\ set$ and the type of command c is $'a \Rightarrow ('s, 'p, 'f)\ com$. All entities formally depend on an auxiliary (logical) variable of type $'a$. The body c formally also depends on this variable, since a nested annotation or loop invariant may also depend on this logical variable. But the raw body without annotations does not depend on the logical variable. The logical variable is only used by the verification condition generator. We express this by defining the whole *specAnno* to be equivalent with the body applied to an arbitrary variable.

The Hoare rule for *specAnno* is mainly an instance of the consequence rule:
$[\![P \subseteq \{s \mid \exists Z.\ s \in P'\ Z \wedge Q'\ Z \subseteq Q \wedge A'\ Z \subseteq A\};\ \forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ c\ Z\ (Q'\ Z),(A'\ Z);\ \forall Z.\ c\ Z = c\ undefined]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ specAnno\ P'\ c\ Q'\ A'\ Q,A$

The side-condition $\forall Z.\ c\ Z = c\ undefined$ expresses the intention of body c explained above: The raw body is independent of the auxiliary variable. This side-condition is solved automatically by the *vcg*. The concrete syntax for this specification annotation is shown in the following example:

lemma (**in** *vars*) $\Gamma\vdash \{\sigma\}$
$\quad$ $\acute{I} :== \acute{M};;$
$\quad$ $ANNO\ \tau.\ \{\!|\tau.\ \acute{I} = {}^{\sigma}M|\!\}$
$\quad\quad$ $\acute{M} :== \acute{N};;\ \acute{N} :== \acute{I}$
$\quad\quad$ $\{\!|\acute{M} = {}^{\tau}N \wedge \acute{N} = {}^{\tau}I|\!\}$
$\quad$ $\{\!|\acute{M} = {}^{\sigma}N \wedge \acute{N} = {}^{\sigma}M|\!\}$

With the annotation we can name an intermediate state τ. Since the postcondition refers to σ we have to link the information about the equivalence of ${}^{\tau}I$ and ${}^{\sigma}M$ in the specification in order to be able to derive the postcondition.

apply *vcg-step*
apply *vcg-step*

1. $\bigwedge\tau.\ \Gamma\vdash \{\!|\tau.\ \acute{I} = {}^{\sigma}M|\!\}\ \acute{M} :== \acute{N};;\ \acute{N} :== \acute{I}\ \{\!|\acute{M} = {}^{\tau}N \wedge \acute{N} = {}^{\tau}I|\!\}$
2. $\Gamma\vdash \{\sigma\}\ \acute{I} :== \acute{M}$
$\quad$ $\{\!|\acute{I} = {}^{\sigma}M \wedge (\forall t.\ {}^{t}M = \acute{N} \wedge {}^{t}N = \acute{I} \longrightarrow \acute{N} = {}^{\sigma}N \wedge \acute{I} = {}^{\sigma}M)|\!\}$

The first subgoal is the isolated Hoare tuple. The second one is the side-condition of the consequence rule that allows us to derive the outermost pre/post condition from our inserted specification. $\acute{I} = {}^{\sigma}M$ is the precondition of the specification, The second conjunct is a simplified version of $\forall t.\ {}^{t}M = \acute{N} \wedge {}^{t}N = \acute{I} \longrightarrow {}^{t}M = {}^{\sigma}N \wedge {}^{t}N = {}^{\sigma}M$ expressing that the postcondition of the specification implies the outermost postcondition.

apply *vcg*

1. $\bigwedge M\ I\ N.\ I = M \Longrightarrow N = N \wedge I = I$
2. $\Gamma \vdash \{\sigma\}\ {}^\acute{}I :== {}^\acute{}M$
$\{\!|{}^\acute{}I = {}^\sigma M \wedge (\forall t.\ {}^t M = {}^\acute{}N \wedge {}^t N = {}^\acute{}I \longrightarrow {}^\acute{}N = {}^\sigma N \wedge {}^\acute{}I = {}^\sigma M)|\!\}$

apply *simp*
apply *vcg*

1. $\bigwedge M\ N.\ M = M \wedge (\forall Ma\ Na.\ Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$

by *simp*

lemma (**in** *vars*)
$\Gamma \vdash \{\sigma\}$
${}^\acute{}I :== {}^\acute{}M;;$
$ANNO\ \tau.\ \{\!|\tau.\ {}^\acute{}I = {}^\sigma M|\!\}$
${}^\acute{}M :== {}^\acute{}N;;\ {}^\acute{}N :== {}^\acute{}I$
$\{\!|{}^\acute{}M = {}^\tau N \wedge {}^\acute{}N = {}^\tau I|\!\}$
$\{\!|{}^\acute{}M = {}^\sigma N \wedge {}^\acute{}N = {}^\sigma M|\!\}$
apply *vcg*

1. $\bigwedge M\ N.\ M = M \wedge (\forall Ma\ Na.\ Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$
2. $\bigwedge M\ I\ N.\ I = M \Longrightarrow N = N \wedge I = I$

by *simp-all*

Note that *vcg-step* changes the order of sequential composition, to allow the user to decompose sequences by repeated calls to *vcg-step*, whereas *vcg* preserves the order.

The above example illustrates how we can introduce a new logical state variable τ. You can introduce multiple variables by using a tuple:

lemma (**in** *vars*)
$\Gamma \vdash \{\sigma\}$
${}^\acute{}I :== {}^\acute{}M;;$
$ANNO\ (n,i,m).\ \{\!|{}^\acute{}I = {}^\sigma M \wedge {}^\acute{}N{=}n \wedge {}^\acute{}I{=}i \wedge {}^\acute{}M{=}m|\!\}$
${}^\acute{}M :== {}^\acute{}N;;\ {}^\acute{}N :== {}^\acute{}I$
$\{\!|{}^\acute{}M = n \wedge {}^\acute{}N = i|\!\}$
$\{\!|{}^\acute{}M = {}^\sigma N \wedge {}^\acute{}N = {}^\sigma M|\!\}$
apply *vcg*

1. $\bigwedge M\ N.\ M = M \wedge (\forall Ma\ Na.\ Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$
2. $\bigwedge M\ I\ N.\ I = M \Longrightarrow N = N \wedge I = I$

by *simp-all*

Lemma Annotations

The specification annotations described before split the verification into several Hoare triples which result in several subgoals. If we instead want to proof the Hoare triples independently as separate lemmas we can use the *LEMMA* annotation to plug together the lemmas. It inserts the lemma in the same fashion as the specification annotation.

lemma (**in** *vars*) *foo-lemma*:
$\forall\, n\ m.\ \Gamma\vdash \{\!|\, ´N = n \wedge ´M = m |\!\}\ ´N :== ´N + 1;;\ ´M :== ´M + 1$
$\{\!|\, ´N = n + 1 \wedge ´M = m + 1 |\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *vars*)
$\Gamma\vdash \{\!|\, ´N = n \wedge ´M = m |\!\}$
LEMMA foo-lemma
$´N :== ´N + 1;;\ ´M :== ´M + 1$
END;;
$´N :== ´N + 1$
$\{\!|\, ´N = n + 2 \wedge ´M = m + 1 |\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *vars*)
$\Gamma\vdash \{\!|\, ´N = n \wedge ´M = m |\!\}$
LEMMA foo-lemma
$´N :== ´N + 1;;\ ´M :== ´M + 1$
END;;
LEMMA foo-lemma
$´N :== ´N + 1;;\ ´M :== ´M + 1$
END
$\{\!|\, ´N = n + 2 \wedge ´M = m + 2 |\!\}$
apply *vcg*
apply *simp*
done

lemma (**in** *vars*)
$\Gamma\vdash \{\!|\, ´N = n \wedge ´M = m |\!\}$
$´N :== ´N + 1;;\ ´M :== ´M + 1;;$
$´N :== ´N + 1;;\ ´M :== ´M + 1$
$\{\!|\, ´N = n + 2 \wedge ´M = m + 2 |\!\}$
apply (*hoare-rule anno*=
LEMMA foo-lemma
$´N :== ´N + 1;;\ ´M :== ´M + 1$
END;;

LEMMA foo-lemma
  ´N :== ´N + 1;; ´M :== ´M + 1
END
in *HoarePartial.annotate-normI*)
apply *vcg*
apply *simp*
done

26.6.3 Total Correctness of Nested Loops

When proving termination of nested loops it is sometimes necessary to express that the loop variable of the outer loop is not modified in the inner loop. To express this one has to fix the value of the outer loop variable before the inner loop and use this value in the invariant of the inner loop. This can be achieved by surrounding the inner while loop with an *ANNO* specification as explained previously. However, this leads to repeating the invariant of the inner loop three times: in the invariant itself and in the the pre- and postcondition of the *ANNO* specification. Moreover one has to deal with the additional subgoal introduced by *ANNO* that expresses how the pre- and postcondition is connected to the invariant. To avoid this extra specification and verification work, we introduce an variant of the annotated while-loop, where one can introduce logical variables by *FIX*. As for the *ANNO* specification multiple logical variables can be introduced via a tuple (*FIX* (*a*,*b*,*c*).).

The Hoare logic rule for the augmented while-loop is a mixture of the invariant rule for loops and the consequence rule for *ANNO*:

$$\llbracket P \subseteq \{s \mid \exists Z.\ s \in I\ Z \wedge (\forall t.\ t \in I\ Z \cap -\ b \longrightarrow t \in Q)\};\ \forall Z\ \sigma.\ \Gamma,\Theta\vdash_{t/F} (\{\sigma\} \cap I\ Z \cap b)\ c\ Z\ (\{t \mid (t, \sigma) \in V\ Z\} \cap I\ Z),A;\ \forall Z.\ c\ Z = c\ \mathit{undefined};\ \forall Z.\ \mathit{wf}\ (V\ Z)\rrbracket \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ \mathit{whileAnnoFix}\ b\ I\ V\ c\ Q,A$$

The first premise expresses that the precondition implies the invariant and that the invariant together with the negated loop condition implies the postcondition. Since both implications may depend on the choice of the auxiliary variable *Z* these two implications are expressed in a single premise and not in two of them as for the usual while rule. The second premise is the preservation of the invariant by the loop body. And the third premise is the side-condition that the computational part of the body does not depend on the auxiliary variable. Finally the last premise is the well-foundedness of the variant. The last two premises are usually discharged automatically by the verification condition generator. Hence usually two subgoals remain for the user, stemming from the first two premises.

The following example illustrates the usage of this rule. The outer loop increments the loop variable *M* while the inner loop increments *N*. To discharge the proof obligation for the termination of the outer loop, we need to know that the inner loop does not mess around with M. This is expressed by introducing the logical variable m and fixing the value of M to it.

lemma (**in** *vars*)
$\Gamma\vdash_t \{\!|\,\acute{}M{=}0 \wedge \acute{}N{=}0|\!\}$
WHILE $(\acute{}M < i)$
INV $\{\!|\,\acute{}M \leq i \wedge (\acute{}M \neq 0 \longrightarrow \acute{}N = j) \wedge \acute{}N \leq j|\!\}$
VAR MEASURE $(i - \acute{}M)$
DO
$\acute{}N :== 0;;$
WHILE $(\acute{}N < j)$
FIX m.
INV $\{\!|\,\acute{}M{=}m \wedge \acute{}N \leq j|\!\}$
VAR MEASURE $(j - \acute{}N)$
DO
$\acute{}N :== \acute{}N + 1$
OD;;
$\acute{}M :== \acute{}M + 1$
OD
$\{\!|\,\acute{}M{=}i \wedge (\acute{}M{\neq}0 \longrightarrow \acute{}N{=}j)|\!\}$
apply *vcg*

1. $\bigwedge M\ N.\ [\![M = 0;\ N = 0]\!] \Longrightarrow M \leq i \wedge (M \neq 0 \longrightarrow N = j) \wedge N \leq j$
2. $\bigwedge M\ N.\ [\![M \leq i;\ M \neq 0 \longrightarrow N = j;\ N \leq j;\ M < i]\!]$
$\Longrightarrow 0 \leq j \wedge$
$(\forall Ma\ N.$
$Ma = M \wedge N \leq j \wedge \neg\ N < j \longrightarrow$
$i - (M + 1) < i - M \wedge M + 1 \leq i \wedge (M + 1 \neq 0 \longrightarrow N$
$= j))$
3. $\bigwedge M\ N.\ [\![N \leq j;\ N < j]\!] \Longrightarrow j - (N + 1) < j - N \wedge M = M \wedge N + 1 \leq j$
4. $\bigwedge M\ N.\ [\![M \leq i;\ M \neq 0 \longrightarrow N = j;\ N \leq j;\ \neg\ M < i]\!]$
$\Longrightarrow M = i \wedge (M \neq 0 \longrightarrow N = j)$

The first subgoal is from the precondition to the invariant of the outer loop. The fourth subgoal is from the invariant together with the negated loop condition of the outer loop to the postcondition. The subgoals two and three are from the body of the outer while loop which is mainly the inner while loop. Because we introduce the logical variable m here, the while Rule described above is used instead of the ordinary while Rule. That is why we end up with two subgoals for the inner loop. Subgoal two is from the invariant and the loop condition of the outer loop to the invariant of the inner loop. And at the same time from the invariant of the inner loop to the invariant of the outer loop (together with the proof obligation that the measure of the outer loop decreases). The universal quantified variables Ma and N are the “fresh” state variables introduced for the final state of the inner loop. The equality $Ma = M$ is the result of the equality $\acute{}M{=}m$ in the inner invariant. Subgoal three is the preservation of the invariant by the inner loop body (together with the proof obligation that the measure of the inner loop decreases).

26.7 Functional Correctness, Termination and Runtime Faults

Total correctness of a program with guards conceptually leads to three verification tasks.

- functional (partial) correctness
- absence of runtime faults
- termination

In case of a modifies specification the functional correctness part can be solved automatically. But the absence of runtime faults and termination may be non trivial. Fortunately the modifies clause is usually just a helpful companion of another specification that expresses the "real" functional behaviour. Therefor the task to prove the absence of runtime faults and termination can be dealt with during the proof of this functional specification. In most cases the absence of runtime faults and termination heavily build on the functional specification parts. So after all there is no reason why we should again prove the absence of runtime faults and termination for the modifies clause. Therefor it suffices to have partial correctness of the modifies clause for a program were all guards are ignored. This leads to the following pattern:

procedures *foo* $(N{::}nat|M{::}nat)$
$\acute{}M :== \acute{}M$
— think of body with guards instead

foo-spec: $\forall\,\sigma.\ \Gamma\vdash_t (P\ \sigma)\ \acute{}M :== PROC\,foo(\acute{}N)\ (Q\ \sigma)$
foo-modifies: $\forall\,\sigma.\ \Gamma\vdash_{/UNIV} \{\sigma\}\ \acute{}M :== PROC\,foo(\acute{}N)$
$\{t.\ t\ may\text{-}only\text{-}modify\text{-}globals\ \sigma\ in\ []\}$

The verification condition generator can solve those modifies clauses automatically and can use them to simplify calls to *foo* even in the context of total correctness.

26.8 Procedures and Locales

Verification of a larger program is organised on the granularity of procedures. We proof the procedures in a bottom up fashion. Of course you can also always use Isabelle's dummy proof *sorry* to prototype your formalisation. So you can write the theory in a bottom up fashion but actually prove the lemmas in any other order.

Here are some explanations of handling of locales. In the examples below, consider $proc_1$ and $proc_2$ to be "leaf" procedures, which do not call any other procedure. Procedure *proc* directly calls $proc_1$ and $proc_2$.

lemma (**in** *proc*$_1$*-impl*) *proc*$_1$*-modifies*:
shows ...

After the proof of *proc*$_1$*-modifies*, the **in** directive stores the lemma in the locale *proc*$_1$*-impl*. When we later on include *proc*$_1$*-impl* or prove another theorem in locale *proc*$_1$*-impl* the lemma *proc*$_1$*-modifies* will already be available as fact.

lemma (**in** *proc*$_1$*-impl*) *proc*$_1$*-spec*:
shows ...

lemma (**in** *proc*$_2$*-impl*) *proc*$_2$*-modifies*:
shows ...

lemma (**in** *proc*$_2$*-impl*) *proc*$_2$*-spec*:
shows ...

lemma (**in** *proc-impl*) *proc-modifies*:
shows ...

Note that we do not explicitly include anything about *proc*$_1$ or *proc*$_2$ here. This is handled automatically. When defining an *impl*-locale it imports all *impl*-locales of procedures that are called in the body. In case of *proc-impl* this means, that *proc*$_1$*-impl* and *proc*$_2$*-impl* are imported. This has the neat effect that all theorems that are proven in *proc*$_1$*-impl* and *proc*$_2$*-impl* are also present in *proc-impl*.

lemma (**in** *proc-impl*) *proc-spec*:
shows ...

As we have seen in this example you only have to prove a procedure in its own *impl* locale. You do not have to include any other locale.

26.9 Records

Before *statespaces* where introduced the state was represented as a *record*. This is still supported. Compared to the flexibility of statespaces there are some drawbacks in particular with respect to modularity. Even names of local variables and parameters are globally visible and records can only be extended in a linear fashion, whereas statespaces also allow multiple inheritance. The usage of records is quite similar to the usage of statespaces. We repeat the example of an append function for heap lists. First we define the global components. Again the appearance of the prefix 'globals' is mandatory. This is the way the syntax layer distinguishes local and global variables.

record *globals-list* =
  next-' :: *ref* ⇒ *ref*
  cont-' :: *ref* ⇒ *nat*

The local variables also have to be defined as a record before the actual definition of the procedure. The parent record *state* defines a generic *globals* field as a place-holder for the record of global components. In contrast to the statespace approach there is no single *locals* slot. The local components are just added to the record.

record $'g$ *list-vars* $=$ $'g$ *state* $+$
  p-' :: *ref*
  q-' :: *ref*
  r-' :: *ref*
  root-' :: *ref*
  tmp-' :: *ref*

Since the parameters and local variables are determined by the record, there are no type annotations or definitions of local variables while defining a procedure.

procedures
  append'(*p*,*q*|*p*) =
    IF ´p=Null THEN ´p :== ´q
    ELSE ´p →´next:== CALL append'(´p→´next,´q) FI

As in the statespace approach, a locale called *append'-impl* is created. Note that we do not give any explicit information which global or local state-record to use. Since the records are already defined we rely on Isabelle's type inference. Dealing with the locale is analogous to the case with statespaces.

lemma (**in** *append'-impl*) *append'-modifies*:
  shows
  $\forall\sigma$. $\Gamma\vdash$ $\{\sigma\}$ *´p :== PROC append'(´p,´q)*
      $\{t.$ *t may-only-modify-globals* σ *in* $[next]\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply (*vcg spec=modifies*)
  done

lemma (**in** *append'-impl*) *append'-spec*:
  shows $\forall\sigma$ *Ps Qs*. $\Gamma\vdash$
      $\{\!|\sigma$. *List ´p ´next Ps* $\wedge$ *List ´q ´next Qs* $\wedge$ *set Ps* $\cap$ *set Qs* $= \{\}|\!\}$
        ´p :== PROC append'(´p,´q)
      $\{\!|$*List ´p ´next (Ps@Qs)* $\wedge$ $(\forall x.\ x \notin$ *set Ps* $\longrightarrow$ *´next x* $=$ ${}^{\sigma}$*next x*$)|\!\}$
  apply (*hoare-rule HoarePartial.ProcRec1*)
  apply *vcg*
  apply *fastforce*
  done

However, in some corner cases the inferred state type in a procedure definition can be too general which raises problems when attempting to proof a suitable specifications in the locale. Consider for example the simple procedure body *´p :== NULL* for a procedure *init*.

procedures *init* (|*p*) =
  ´p:== Null

Here Isabelle can only infer the local variable record. Since no reference to any global variable is made the type fixed for the global variables (in the locale *init'-impl*) is a type variable say $'g$ and not a *globals-list* record. Any specification mentioning *next* or *cont* restricts the state type and cannot be added to the locale *init-impl*. Hence we have to restrict the body ´*p* :== *NULL* in the first place by adding a typing annotation:

procedures *init'* (|*p*) =
 ´*p*:== *Null*::(('*a globals-list-scheme*, '*b*) *list-vars-scheme*, *char list*, '*c*) *com*

26.9.1 Extending State Spaces

The records in Isabelle are extensible [7, 6]. In principle this can be exploited during verification. The state space can be extended while we we add procedures. But there is one major drawback:

- records can only be extended in a linear fashion (there is no multiple inheritance)

You can extend both the main state record as well as the record for the global variables.

26.9.2 Mapping Variables to Record Fields

Generally the state space (global and local variables) is flat and all components are accessible from everywhere. Locality or globality of variables is achieved by the proper *init* and *return*/*result* functions in procedure calls. What is the best way to map programming language variables to the state records? One way is to disambiguate all names, by using the procedure names as prefix or the structure names for heap components. This leads to long names and lots of record components. But for local variables this is not necessary, since variable i of procedure A and variable i of procedure B can be mapped to the same record component, without any harm, provided they have the same logical type. Therefor for local variables it is preferable to map them per type. You only have to distinguish a variable with the same name if they have a different type. Note that all pointers just have logical type *ref*. So you even do not have to distinguish between a pointer p to a integer and a pointer p to a list. For global components (global variables and heap structures) you have to disambiguate the name. But hopefully the field names of structures have different names anyway. Also note that there is no notion of hiding of a global component by a local one in the logic. You have to disambiguate global and local names! As the names of the components show up in the specifications and the proof obligations, names are even more important as for programming. Try to find meaningful and short names, to avoid cluttering up your reasoning.

References

[1] E. Alkassar, N. Schirmer, and A. Starostin. Formal pervasive verification of a paging mechanism. In *14th intl Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS08), to appear*, LNCS. Springer, 2008.

[2] C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs: International Workshop, TYPES 2003, Torino, Italy, April 30–May 4, 2003, Selected Papers*, number 3085 in LNCS, pages 34–50. Springer, 2004.

[3] P. V. Homeier. *Trustworthy Tools for Trustworthy Programs: A Mechanically Verified Verification Condition Generator for the Total Correctness of Procedures.* PhD thesis, Department of Computer Science, University of California, Los Angeles, 1995.

[4] D. Leinenbach and E. Petrova. Pervasive compiler verification – from verified programs to verified systems. In *3rd intl Workshop on Systems Software Verification (SSV08), to appear*. Elsevier Science B. V., 2008.

[5] F. Mehta and T. Nipkow. Proving pointer programs in higher-order logic. In F. Baader, editor, *Automated Deduction — CADE-19*, volume 2741 of *LNCS*, pages 121–135. Springer, 2003.

[6] W. Naraschewski and M. Wenzel. Object-oriented verification based on record subtyping in higher-order logic. In *Theorem Proving in Higher Order Logics: 11th International Conference, TPHOLs'98*, volume 1479 of *LNCS*. Springer, 1998.

[7] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. http://www.in.tum.de/~nipkow/LNCS2283/.

[8] V. Ortner and N. Schirmer. Verification of BDD normalization. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 2005*, volume 3603 of *LNCS*, pages 261–277. Springer, 2005.

[9] N. Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL.* PhD thesis, Technische Universität München, 2006. Available from http://mediatum2.ub.tum.de/doc/601799/601799.pdf.

[10] H. Tuch, G. Klein, and M. Norrish. Types, bytes, and separation logic. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 97–108, New York, NY, USA, 2007. ACM Press.