TimSort

By zy

July 1, 2018

Contents

1 Implementation of integer numbers by target-language integers

theory *Code-Target-Int*
imports *Main*
begin

code-datatype *int-of-integer*

declare [[*code drop*: *integer-of-int*]]

context
includes *integer.lifting*
begin

lemma [*code*]:
integer-of-int (*int-of-integer* *k*) = *k*
by *transfer rule*

lemma [*code*]:
Int.Pos = *int-of-integer* ∘ *integer-of-num*
by *transfer* (*simp add*: *fun-eq-iff*)

lemma [*code*]:
Int.Neg = *int-of-integer* ∘ *uminus* ∘ *integer-of-num*
by *transfer* (*simp add*: *fun-eq-iff*)

lemma [*code-abbrev*]:
int-of-integer (*numeral* *k*) = *Int.Pos* *k*
by *transfer simp*

lemma [*code-abbrev*]:
int-of-integer (− *numeral* *k*) = *Int.Neg* *k*
by *transfer simp*

context
begin

qualified definition *positive* :: *num* ⇒ *int*
where [*simp*]: *positive* = *numeral*

qualified definition *negative* :: *num* ⇒ *int*
where [*simp*]: *negative* = *uminus* ∘ *numeral*

lemma [*code-computation-unfold*]:
numeral = *positive*
Int.Pos = *positive*
Int.Neg = *negative*
by (*simp-all add*: *fun-eq-iff*)

end

lemma [*code*, *symmetric*, *code-post*]:
0 = *int-of-integer* *0*
by *transfer simp*

lemma [*code*, *symmetric*, *code-post*]:
1 = *int-of-integer* *1*
by *transfer simp*

lemma [*code-post*]:
int-of-integer (− *1*) = − *1*
by *simp*

lemma [*code*]:
$k + l = \textit{int-of-integer}\ (\textit{of-int}\ k + \textit{of-int}\ l)$
by *transfer simp*

lemma [*code*]:
$-\ k = \textit{int-of-integer}\ (-\ \textit{of-int}\ k)$
by *transfer simp*

lemma [*code*]:
$k - l = \textit{int-of-integer}\ (\textit{of-int}\ k - \textit{of-int}\ l)$
by *transfer simp*

lemma [*code*]:
$\textit{Int.dup}\ k = \textit{int-of-integer}\ (\textit{Code-Numeral.dup}\ (\textit{of-int}\ k))$
by *transfer simp*

declare [[*code drop*: *Int.sub*]]

lemma [*code*]:
$k * l = \textit{int-of-integer}\ (\textit{of-int}\ k * \textit{of-int}\ l)$
by *simp*

lemma [*code*]:
$k\ \textit{div}\ l = \textit{int-of-integer}\ (\textit{of-int}\ k\ \textit{div}\ \textit{of-int}\ l)$
by *simp*

lemma [*code*]:
$k\ \textit{mod}\ l = \textit{int-of-integer}\ (\textit{of-int}\ k\ \textit{mod}\ \textit{of-int}\ l)$
by *simp*

lemma [*code*]:
$\textit{divmod}\ m\ n = \textit{map-prod}\ \textit{int-of-integer}\ \textit{int-of-integer}\ (\textit{divmod}\ m\ n)$
unfolding *prod-eq-iff divmod-def map-prod-def case-prod-beta fst-conv snd-conv*
by *transfer simp*

lemma [*code*]:
$\textit{HOL.equal}\ k\ l = \textit{HOL.equal}\ (\textit{of-int}\ k :: \textit{integer})\ (\textit{of-int}\ l)$
by *transfer* (*simp add*: *equal*)

lemma [*code*]:
$k \leq l \longleftrightarrow (\textit{of-int}\ k :: \textit{integer}) \leq \textit{of-int}\ l$
by *transfer rule*

lemma [*code*]:
$k < l \longleftrightarrow (\textit{of-int}\ k :: \textit{integer}) < \textit{of-int}\ l$
by *transfer rule*

declare [[*code drop*: $\textit{gcd} :: \textit{int} \Rightarrow$ - $\textit{lcm} :: \textit{int} \Rightarrow$ -]]

lemma *gcd-int-of-integer* [*code*]:
gcd (*int-of-integer x*) (*int-of-integer y*) = *int-of-integer* (*gcd x y*)
by *transfer rule*

lemma *lcm-int-of-integer* [*code*]:
lcm (*int-of-integer x*) (*int-of-integer y*) = *int-of-integer* (*lcm x y*)
by *transfer rule*

end

lemma (**in** *ring-1*) *of-int-code-if*:
of-int k = (*if k* = *0 then 0*
else if k < *0 then* − *of-int* (− *k*)
else let
l = *2* ∗ *of-int* (*k div 2*);
j = *k mod 2*
in if j = *0 then l else l* + *1*)
proof −
from *div-mult-mod-eq* **have** ∗: *of-int k* = *of-int* (*k div 2* ∗ *2* + *k mod 2*) **by** *simp*
show *?thesis*
by (*simp add*: *Let-def of-int-add* [*symmetric*]) (*simp add*: ∗ *mult.commute*)
qed

declare *of-int-code-if* [*code*]

lemma [*code*]:
nat = *nat-of-integer* ∘ *of-int*
including *integer.lifting* **by** *transfer* (*simp add*: *fun-eq-iff*)

code-identifier
code-module *Code-Target-Int* ⇀
(*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

end

2 Avoidance of pattern matching on natural numbers

theory *Code-Abstract-Nat*
imports *Main*
begin

When natural numbers are implemented in another than the conventional inductive *0*/*Suc* representation, it is necessary to avoid all pattern matching on natural numbers altogether. This is accomplished by this theory (up to a certain extent).

2.1 Case analysis

Case analysis on natural numbers is rephrased using a conditional expression:

lemma [*code*, *code-unfold*]:
  case-nat = ($\lambda f\ g\ n.$ *if* $n = 0$ *then* *f* *else* *g* $(n - 1)$)
  by (*auto simp add*: *fun-eq-iff dest*!: *gr0-implies-Suc*)

2.2 Preprocessors

The term *Suc n* is no longer a valid pattern. Therefore, all occurrences of this term in a position where a pattern is expected (i.e. on the left-hand side of a code equation) must be eliminated. This can be accomplished – as far as possible – by applying the following transformation rule:

lemma *Suc-if-eq*:
  assumes $\bigwedge n.\ f\ (Suc\ n) \equiv h\ n$
  assumes $f\ 0 \equiv g$
  shows $f\ n \equiv$ *if* $n = 0$ *then* *g* *else* *h* $(n - 1)$
  by (*rule eq-reflection*) (*cases n*, *insert assms*, *simp-all*)

The rule above is built into a preprocessor that is plugged into the code generator.

setup ‹

```
let

val Suc-if-eq = Thm.incr-indexes 1 @{thm Suc-if-eq};

fun remove-suc ctxt thms =
  let
    val vname = singleton (Name.variant-list (map fst
      (fold (Term.add-var-names o Thm.full-prop-of) thms []))) n;
    val cv = Thm.cterm-of ctxt (Var ((vname, 0), HOLogic.natT));
    val lhs-of = snd o Thm.dest-comb o fst o Thm.dest-comb o Thm.cprop-of;
    val rhs-of = snd o Thm.dest-comb o Thm.cprop-of;
    fun find-vars ct = (case Thm.term-of ct of
       (Const (@{const-name Suc}, -) $ Var -) => [(cv, snd (Thm.dest-comb ct))]
     | - $ - =>
       let val (ct1, ct2) = Thm.dest-comb ct
       in
         map (apfst (fn ct => Thm.apply ct ct2)) (find-vars ct1) @
         map (apfst (Thm.apply ct1)) (find-vars ct2)
       end
     | - => []);
    val eqs = maps
      (fn thm => map (pair thm) (find-vars (lhs-of thm))) thms;
    fun mk-thms (thm, (ct, cv')) =
      let
        val thm' =
```

```
      Thm.implies-elim
       (Conv.fconv-rule (Thm.beta-conversion true)
         (Thm.instantiate'
           [SOME (Thm.ctyp-of-cterm ct)] [SOME (Thm.lambda cv ct),
             SOME (Thm.lambda cv' (rhs-of thm)), NONE, SOME cv']
           Suc-if-eq)) (Thm.forall-intr cv' thm)
    in
      case map-filter (fn thm'' =>
          SOME (thm'', singleton
            (Variable.trade (K (fn [thm'''] => [thm''' RS thm']))
              (Variable.declare-thm thm'' ctxt)) thm'')
        handle THM - => NONE) thms of
          [] => NONE
        | thmps =>
            let val (thms1, thms2) = split-list thmps
            in SOME (subtract Thm.eq-thm (thm :: thms1) thms @ thms2) end
    end
  in get-first mk-thms eqs end;

fun eqn-suc-base-preproc ctxt thms =
  let
    val dest = fst o Logic.dest-equals o Thm.prop-of;
    val contains-suc = exists-Const (fn (c, -) => c = @{const-name Suc});
  in
    if forall (can dest) thms andalso exists (contains-suc o dest) thms
    then thms |> perhaps-loop (remove-suc ctxt) |> (Option.map o map) Drule.zero-var-indexes
      else NONE
  end;

val eqn-suc-preproc = Code-Preproc.simple-functrans eqn-suc-base-preproc;

in

  Code-Preproc.add-functrans (eqn-Suc, eqn-suc-preproc)

end;
〉
```

end

3 Implementation of natural numbers by target-language integers

theory *Code-Target-Nat*
imports *Code-Abstract-Nat*
begin

3.1 Implementation for *nat*

context
includes *natural.lifting integer.lifting*
begin

lift-definition *Nat* :: *integer* ⇒ *nat*
  is *nat*
  .

lemma [*code-post*]:
  Nat 0 = *0*
  Nat 1 = *1*
  Nat (*numeral k*) = *numeral k*
  by (*transfer*, *simp*)+

lemma [*code-abbrev*]:
  integer-of-nat = *of-nat*
  by *transfer rule*

lemma [*code-unfold*]:
  Int.nat (*int-of-integer k*) = *nat-of-integer k*
  by *transfer rule*

lemma [*code abstype*]:
  Code-Target-Nat.Nat (*integer-of-nat n*) = *n*
  by *transfer simp*

lemma [*code abstract*]:
  integer-of-nat (*nat-of-integer k*) = *max 0 k*
  by *transfer auto*

lemma [*code-abbrev*]:
  nat-of-integer (*numeral k*) = *nat-of-num k*
  by *transfer* (*simp add*: *nat-of-num-numeral*)

context
begin

qualified definition *natural* :: *num* ⇒ *nat*
  where [*simp*]: *natural* = *nat-of-num*

lemma [*code-computation-unfold*]:
  numeral = *natural*
  nat-of-num = *natural*
  by (*simp-all add*: *nat-of-num-numeral*)

end

lemma [*code abstract*]:

integer-of-nat (*nat-of-num n*) = *integer-of-num n*
by (*simp add*: *nat-of-num-numeral integer-of-nat-numeral*)

lemma [*code abstract*]:
integer-of-nat 0 = *0*
by *transfer simp*

lemma [*code abstract*]:
integer-of-nat 1 = *1*
by *transfer simp*

lemma [*code*]:
Suc n = *n* + *1*
by *simp*

lemma [*code abstract*]:
integer-of-nat (*m* + *n*) = *of-nat m* + *of-nat n*
by *transfer simp*

lemma [*code abstract*]:
integer-of-nat (*m* − *n*) = *max 0* (*of-nat m* − *of-nat n*)
by *transfer simp*

lemma [*code abstract*]:
integer-of-nat (*m* ∗ *n*) = *of-nat m* ∗ *of-nat n*
by *transfer* (*simp add*: *of-nat-mult*)

lemma [*code abstract*]:
integer-of-nat (*m div n*) = *of-nat m div of-nat n*
by *transfer* (*simp add*: *zdiv-int*)

lemma [*code abstract*]:
integer-of-nat (*m mod n*) = *of-nat m mod of-nat n*
by *transfer* (*simp add*: *zmod-int*)

lemma [*code*]:
Divides.divmod-nat m n = (*m div n*, *m mod n*)
by (*fact divmod-nat-div-mod*)

lemma [*code*]:
divmod m n = *map-prod nat-of-integer nat-of-integer* (*divmod m n*)
by (*simp only*: *prod-eq-iff divmod-def map-prod-def case-prod-beta fst-conv snd-conv*)
(*transfer*, *simp-all only*: *nat-div-distrib nat-mod-distrib*
zero-le-numeral nat-numeral)

lemma [*code*]:
HOL.equal m n = *HOL.equal* (*of-nat m* :: *integer*) (*of-nat n*)
by *transfer* (*simp add*: *equal*)

lemma [*code*]:
$m \le n \longleftrightarrow$ (*of-nat* m :: *integer*) $\le$ *of-nat* n
by *simp*

lemma [*code*]:
$m < n \longleftrightarrow$ (*of-nat* m :: *integer*) $<$ *of-nat* n
by *simp*

lemma *num-of-nat-code* [*code*]:
num-of-nat = *num-of-integer* $\circ$ *of-nat*
by *transfer* (*simp add*: *fun-eq-iff*)

end

lemma (**in** *semiring-1*) *of-nat-code-if*:
of-nat n = (*if* $n = 0$ *then* 0
else let
(m, q) = *Divides.divmod-nat* n 2;
m' = 2 $*$ *of-nat* m
in if $q = 0$ *then* m' *else* $m' + 1$)
proof −
from *div-mult-mod-eq* **have** $*$: *of-nat* n = *of-nat* (n *div* 2 $*$ 2 + n *mod* 2) **by** *simp*
show *?thesis*
by (*simp add*: *Let-def divmod-nat-div-mod of-nat-add* [*symmetric*])
(*simp add*: $*$ *mult.commute of-nat-mult add.commute*)
qed

declare *of-nat-code-if* [*code*]

definition *int-of-nat* :: *nat* $\Rightarrow$ *int* **where**
[*code-abbrev*]: *int-of-nat* = *of-nat*

lemma [*code*]:
int-of-nat n = *int-of-integer* (*of-nat* n)
by (*simp add*: *int-of-nat-def*)

lemma [*code abstract*]:
integer-of-nat (*nat* k) = *max* 0 (*integer-of-int* k)
including *integer.lifting* **by** *transfer auto*

lemma *term-of-nat-code* [*code*]:
— Use *nat-of-integer* in term reconstruction instead of *Code-Target-Nat.Nat* such that reconstructed terms can be fed back to the code generator
term-of-class.term-of n =
Code-Evaluation.App
(*Code-Evaluation.Const* (*STR* ''*Code-Numeral.nat-of-integer*'')
(*typerep.Typerep* (*STR* ''*fun*'')
[*typerep.Typerep* (*STR* ''*Code-Numeral.integer*'') [],

typerep.Typerep (*STR ''Nat.nat''*) []]))
(*term-of-class.term-of* (*integer-of-nat n*))
by (*simp add*: *term-of-anything*)

lemma *nat-of-integer-code-post* [*code-post*]:
nat-of-integer 0 = *0*
nat-of-integer 1 = *1*
nat-of-integer (*numeral k*) = *numeral k*
including *integer.lifting* **by** (*transfer*, *simp*)+

code-identifier
code-module *Code-Target-Nat* ⇀
(*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

end

4 Implementation of natural and integer numbers by target-language integers

theory *Code-Target-Numeral*
imports *Code-Target-Int Code-Target-Nat*
begin

end
theory *TimSortLemma*
imports *Main ~~/src/HOL/Library/Code-Target-Numeral*
begin
definition *list-copy* :: *'a list* ⇒ *nat* ⇒ *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a list* **where**
list-copy xs n ys m l = (*take n xs*) @ (*take l* (*drop m ys*)) @ (*drop* (*n*+*l*) *xs*)

value [*1*,*2*,*3*,*4*,*5*::*int*]

value *let a* = [*1*::*nat*,*2*,*3*,*4*,*5*] *in list-copy a 1 a 0 5*

lemma *list-copy-front*:*n*<*length xs* ∧ *m*<*length ys* ∧ (*m*+*l*)≤*length ys* ⟹ *take n* (*list-copy xs n ys m l*) = *take n xs*
by (*simp add*:*list-copy-def*)

lemma *list-copy-middle*:*n*<*length xs* & *m*<*length ys*& (*m*+*l*)≤*length ys* ⟹
take l (*drop n* (*list-copy xs n ys m l*)) = *take l* (*drop m ys*)
by (*auto simp add*:*list-copy-def*)

lemma *list-copy-end*:*n*<*length xs* & *m*<*length ys*& (*m*+*l*)≤*length ys* ⟹ *drop* (*n*+*l*) (*list-copy xs n ys m l*) = *drop* (*n*+*l*) *xs*
apply (*auto simp add*:*list-copy-def*)
apply (*metis diff-add-inverse2 le-add-diff-inverse less-imp-le-nat min.absorb2 nat-add-left-cancel-le*)

```
  done

lemma list-copy-len[simp]:(m+l)≤length ys ⟹ (n+l)≤length xs ⟹ (length (list-copy
xs n ys m l) = length xs)
  by (auto simp add:list-copy-def)

lemma list-copy-zero:list-copy xs n ys m 0 = xs
  by (simp add:list-copy-def)

definition sorted-in::int list ⇒ nat ⇒ nat ⇒ bool where
sorted-in xs lo hi = (∀ i. (i≥lo∧i<hi)⟶(xs!i≤xs!(i+1)))

thm allE
value sorted [0,−1::int]
value ([0::int,−1]!0) ≤([0::int,−1]!1)
lemma sorted-in-one-more: sorted-in xs lo hi ⟹ sorted-in (x#xs) (Suc lo) (Suc
hi)
  apply (auto simp add:sorted-in-def)
  apply (erule-tac ?x = i−1 in allE)
  apply auto
  done

lemma sorted-in-conca:sorted-in xs lo mid ∧ sorted-in xs mid hi ⟹ sorted-in xs
lo hi
  apply (auto simp add:sorted-in-def)
  using not-less by blast

lemma sorted-in-hi:sorted-in xs lo hi ∧ xs!hi<xs!(hi+1) ⟹ sorted-in xs lo (hi+1)
  apply (auto simp add:sorted-in-def)
  using less-antisym by fastforce

lemma sorted-in-pick-two:sorted-in xs lo hi ∧ i≥lo ∧ j≤hi ∧ i≤j ⟹ xs!i ≤ xs!j
  apply (simp add:sorted-in-def)
  apply (induct j arbitrary:xs lo hi i)
   apply simp
  apply (case-tac i=Suc j)
   apply simp
  apply (subgoal-tac xs ! i ≤ xs ! j)
  apply (meson Suc-le-lessD dual-order.trans le-SucE)
  by (meson Suc-leD le-SucE)

lemma le-half:a<(b::nat) ⟹ (a+b) div 2 < b
proof −
  assume le:a<b
  from this have a+b<b+b by simp
  from this have (a+b) div 2 < (b+b) div 2
    using div-le-mono by auto
  from this show ?thesis
    by linarith
```

qed

lemma *conca-nth-a*[*simp*]: $i < length\ xs \Longrightarrow (xs@ys)!i = xs!i$
  using *nth-take*[*of i length xs xs@ys*] **by** *auto*
lemma *conca-nth-b*[*simp*]: $i \geq length\ xs \Longrightarrow (xs@ys)!i = ys!(i - length\ xs)$
  by (*simp add*: *nth-append*)
lemma *list-copy-i-front*[*simp*]:$(n+l) \leq length\ xs \Longrightarrow (m+l) \leq length\ ys \Longrightarrow i<n \Longrightarrow$ $(list\text{-}copy\ xs\ n\ ys\ m\ l)!i = xs!i$
  apply (*auto simp add*:*list-copy-def*)
  done
lemma *list-copy-i-mid*[*simp*]:$(n+l) \leq length\ xs \Longrightarrow (m+l) \leq length\ ys \Longrightarrow i \geq n \wedge i<(n+l)$ $\Longrightarrow (list\text{-}copy\ xs\ n\ ys\ m\ l)!i = ys!(i-n+m)$
  apply (*auto simp add*:*list-copy-def*)
  apply (*subgoal-tac min* (*length xs*) $n = n$)
   apply (*simp*)
   apply (*subgoal-tac* $i-n < l$)
    apply (*auto simp add*:*add.commute*)
  done

lemma *list-copy-i-end*[*simp*]:$(n+l) \leq length\ xs \Longrightarrow (m+l) \leq length\ ys \Longrightarrow i \geq n+l \wedge i<length$ $xs \Longrightarrow (list\text{-}copy\ xs\ n\ ys\ m\ l)!i = xs!i$
  apply (*auto simp add*:*list-copy-def*)
  apply (*subgoal-tac min* (*length xs*) n + *min* (*length ys* $- m$) $l = n+l$)
   apply *auto*
  done
lemma *length-take-one*:$n \leq length\ xs \Longrightarrow length\ (take\ n\ xs) = n$
  by *auto*

definition *elem-bigger-than-next-2*::*nat list* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
  where *elem-bigger-than-next-2 array index* $\equiv$
    $(index+2<(size\ array)) \longrightarrow$
    $(array!index) > (array!(index+1)) + (array!(index+2))$

definition *elem-bigger-than-next*::*nat list* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
  where *elem-bigger-than-next array index* $\equiv$
    $(index+1<(size\ array)) \longrightarrow$
    $(array!index) > (array!(index+1))$

definition *elem-larger-than-bound*::*nat list* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
  where *elem-larger-than-bound array index bound* $\equiv$
    $(index<(size\ array)) \longrightarrow (array!index) \geq bound$

definition *elem-inv*::*nat list* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
  where *elem-inv array index bound* $\equiv$
    (*elem-bigger-than-next-2 array index*) $\wedge$
    (*elem-bigger-than-next array index*) $\wedge$
    (*elem-larger-than-bound array index bound*)

```
value (1::int)#2#3#[]
value last ((1::int)#2#3#[])
value butlast ((1::int)#2#3#[])

value ((1::int)#2#3#[])!2
value take 2 ((1::int)#2#3#[])
value ((1::int)#2#3#[])[2:=10]
value replicate 5 (6::nat)
value if (3::nat)>4 then 5::nat else (if (3::nat)>4 then 7 else 8)

value if 150 < (120::nat) then (4::nat) else
          (if (150::nat) < 1542 then (9::nat) else
           (if (150::nat) < 119151 then (18::nat) else (39::nat)))

lemma suc-simp:Suc n = n+1
  by simp

primrec sum :: nat list ⇒ nat
  where
sum [] = 0 |
sum (x#xs) = x+(sum xs)

primrec sumn :: nat list ⇒ nat ⇒ nat
  where
sumn a 0 = 0 |
sumn a (Suc n) = a!n + (sumn a n)

value sumn (1#2#3#[]) 2

fun fib:: nat ⇒ nat where
fib 0 = 1 |
fib (Suc 0) = 1 |
fib (Suc (Suc n)) = fib(n) + fib(Suc n)


fun fib2:: nat ⇒ nat where
fib2 0 = 0 |
fib2 (Suc 0) = 1 |
fib2 (Suc (Suc n)) = fib2(n) + fib2(Suc n) + 1

lemma fib-plus-2: fib(n+2) = fib(n+1) + fib(n)
  by auto

lemma fib2-plus-2: fib2(n+2) = fib2(n+1) + fib2(n) + 1
  by auto

value ((fib 5) − 1)*16 + (fib2 5) − (5)
value ((fib 19) − 1)*16 + (fib2 19) − (19)
```

value *fib 3*
value *fib2 3*
term *15::nat*

lemma *less-than*: $\llbracket(a::nat)\leq(b::nat);\ a\neq b\rrbracket \Longrightarrow a<b$
  by *simp*

lemma *sum-append-one*: $sum(a@[(b::nat)]) = sum\ a + b$
  apply (*induction a*)
   apply *auto*
  done

lemma *accu-add*: $\llbracket\forall\, i<(n-1).\ a!i + b!i = a!(i+1); size\ a = size\ b;\ size\ a \geq n;\ size\ a > 0;\ n > 0\rrbracket \Longrightarrow$
  $a!(n-1)+b!(n-1) = a!(0) + (sum\ (take\ n\ b))$
  apply (*induction n*)
   apply *auto*
  apply (*case-tac n=0*)
   apply (*auto simp add*: *take-Suc-conv-app-nth*)
  using *sum-append-one* **by** *auto*

lemma *list-take-and-drop*: $xs = take\ n\ xs\ @\ drop\ n\ xs$
  by *auto*

lemma *sumn-update-no-use*: $m\leq length\ a \Longrightarrow n\geq m \Longrightarrow sumn\ (a[n:=t])\ m = sumn\ a\ m$
  apply (*induct m arbitrary*: *a t n*)
   apply (*auto*)
  done

lemma *sumn1*:$n\geq 2 \Longrightarrow n\leq length\ a \Longrightarrow sumn\ (a[n - 2 := a\ !\ (n - 2) + a\ !\ Suc\ (n - 2)])\ (n - Suc\ 0) = sumn\ a\ n$
  apply (*case-tac n*)
   apply *simp-all*
  apply (*case-tac nat*)
   apply (*simp-all add*:*sumn-update-no-use*)
  done

lemma *sumn2*: $n\geq 3 \Longrightarrow n \leq length\ a \Longrightarrow sumn\ (a[n - 3 := a\ !\ (n - 3) + a\ !\ Suc\ (n - 3),$
    $Suc\ (n - 3) := a\ !\ Suc\ (Suc\ (n - 3))])\ (n - Suc\ 0) = sumn\ a\ n$
  apply (*case-tac n*)
   apply *simp-all*

apply (*case-tac nat*)
 apply (*simp-all*)
apply (*case-tac nata*)
 apply (*simp-all*)
apply (*simp-all add:sumn-update-no-use*)
done

lemma *nth-list-update-neq2*: $i \neq j \Longrightarrow k \neq j \Longrightarrow xs[i{:=}x,k{:=}y]!j = xs!j$
apply (*induct xs arbitrary:i j k*)
 apply (*auto simp add: nth-Cons split: nat.split*)
done

lemma *run-len-iter*: $\forall\, i<l - Suc\ 0.\ ys\ !\ i + xs\ !\ i = ys\ !\ Suc\ i \Longrightarrow$
$l > 0 \Longrightarrow$
$l \leq size\ xs \Longrightarrow$
$size\ xs = size\ ys \Longrightarrow$
$ys!0 + sumn\ xs\ l = xs!(l-1) + ys!(l-1)$
proof (*induction l arbitrary: xs ys*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc l*)
from *Suc.IH Suc.prems* **show** *?case*
proof (*cases l=0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
assume *a0*:$length\ xs = length\ ys$ **and** *a1*:$Suc\ l \leq length\ xs$ **and** *a2*:$0 < Suc\ l$ **and** *a3*:$l \neq 0$ **and** *a4*: $\forall\, i<Suc\ l - Suc\ 0.\ ys\ !\ i + xs\ !\ i = ys\ !\ Suc\ i$ **and**
a5: $(\bigwedge ys\ xs.\ \forall\, i<l - Suc\ 0.\ ys\ !\ i + xs\ !\ i = ys\ !\ Suc\ i \Longrightarrow 0 < l \Longrightarrow l \leq length\ xs \Longrightarrow length\ xs = length\ ys \Longrightarrow ys\ !\ 0 + sumn\ xs\ l = xs\ !\ (l - 1) + ys\ !\ (l - 1))$
from *this* **have** *step0*:$ys\ !\ 0 + sumn\ xs\ l = xs\ !\ (l - 1) + ys\ !\ (l - 1)$
by (*metis Suc-diff-Suc Suc-diff-diff Suc-leD Suc-pred less-SucI not-gr-zero*)
from *a2 a3 a4* **have** *one-run*:$ys!(l-1) + xs!(l-1) = ys!l$
by (*metis One-nat-def Suc-pred' diff-Suc-1 diff-Suc-less less-antisym*)
from *sumn.simps*(*2*) **have** *one-sumn*:$sumn\ xs\ (l+1) = sumn\ xs\ l + xs!l$ **by** *simp*
from *step0 one-run one-sumn* **show** *?thesis* **by** *simp*
qed
qed

lemma *rl-elem-lower-bound*: $\forall\, i.\ 3\leq i \land i\leq l \longrightarrow (l-i < l-2 \longrightarrow rl!Suc\ (l-i) + rl!Suc\ (Suc\ (l-i)) < rl!(l-i)) \land (l-i < l-1 \longrightarrow rl!Suc\ (l-i) < rl!(l-i)) \land u \leq rl!(l-i) \Longrightarrow$
$rl!(l-1) < rl!(l-2) \Longrightarrow u \leq rl!(l-1) \Longrightarrow length\ rl = l \Longrightarrow l\geq 2 \Longrightarrow k<l$

$\Longrightarrow$ *rl*!(*l*−*1*−*k*) ≥ *u*∗(*fib k*) + (*fib2 k*)
proof (*induction k arbitrary*:*u rl l rule*:*fib2*.*induct*)
  case *1*
  then show *?case* **by** *auto*
next
  case *2*
  then have *rl*!(*l*−*2*)≥*u*+*1* **by** *simp*
  then show *?case*
    by (*metis One-nat-def diff-diff-left fib*.*simps*(*2*) *fib2*.*simps*(*2*) *nat-mult-1-right one-add-one*)
next
  case (*3 n*)
  from *this* **have** *n1*:*u* ∗ *fib n* + *fib2 n* ≤ *rl* ! (*l* − *1* − *n*) **and** *n2*:*u* ∗ *fib* (*Suc n*) + *fib2* (*Suc n*) ≤ *rl* ! (*l* − *1* − *Suc n*)
    by (*metis Suc-lessD*)+
  from *3*.*prems* **have** *larger-than-next-two*:*rl* ! (*l* − *1* − *n*) + *rl* ! (*l* − *1* − *Suc n*) < *rl* ! (*l* − *1* − *Suc* (*Suc n*))
    apply(*drule-tac x*=*n*+*3* **in** *spec*)
    apply (*clarsimp*)
    apply (*subgoal-tac l* − (*n* + *3*) < *l* − *2*)
     prefer *2*
     apply *linarith*
   by (*smt Suc-diff-Suc Suc-lessD add*.*commute add-2-eq-Suc′ add-Suc-right nat-le-linear not-less numeral-2-eq-2 numeral-3-eq-3*)
  from *n1 n2 larger-than-next-two* **show** *?case* **by** (*simp add*:*distrib-left*)
qed

lemma *append-suc*: *n*≥*1* $\Longrightarrow$ *length xs* ≥ *n* $\Longrightarrow$ (*x*#*xs*)!*n* = *xs*!(*n*−*1*)
proof (*induction n arbitrary*:*xs x*)
case *0*
  then show *?case* **by** *simp*
next
  case (*Suc n*)
  then show *?case* **by** *force*
qed

lemma *minus-same-num*: *a*=*b* $\Longrightarrow$ *a*−*c* = *b*−*c* **by** *simp*

lemma *minus-suc-plus-one*: *a* + *1* − (*Suc b*) = *a* − *b* **by** *simp*

lemma *minus-exc*: *a*≥*c* $\Longrightarrow$ (*a*::*nat*)+*b*−*c* = *a*−*c*+*b* **by** *simp*

lemma *fib2-1*: *fib2 n* ≥ *n*
  apply (*induction n rule*:*fib2*.*induct*)
    apply *auto*
  done

lemma *fib-1*:*fib n* ≥ *Suc 0*
  apply (*induction n rule*:*fib*.*induct*)

apply *auto*
done

lemma *sumn-first-one-out*:$l \le$ *length* $(a\#rl) \Longrightarrow l > 0 \Longrightarrow$ *sumn* $(a \# rl)\ l = a + $ *sumn rl* $(l - Suc\ 0)$
proof (*induction l arbitrary*: *a rl*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc l*)
then show *?case*
proof (*cases* $l=0$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis* **using** *Suc.IH*[*of a rl*] *Suc.prems*
by (*smt Suc-leD Suc-pred ab-semigroup-add-class.add-ac*(*1*) *add.commute diff-Suc-1 less-Suc-eq nth-Cons-pos sumn.simps*(*2*))
qed
qed

lemma *rl-sum-lower-bound*: $\forall i.\ 3 \le i \wedge i \le l \longrightarrow (l-i < l-2 \longrightarrow rl!Suc\ (l-i) + rl!Suc\ (Suc\ (l-i)) < rl!(l-i)) \wedge (l-i < l-1 \longrightarrow rl!Suc\ (l-i) < rl!(l-i)) \wedge u \le rl!(l-i) \Longrightarrow$
$rl!(l-1) < rl!(l-2) \Longrightarrow u \le rl!(l-1) \Longrightarrow$ *length* $rl = l \Longrightarrow l \ge 2$
$\Longrightarrow$ *sumn rl* $l \ge u*((fib\ (l+1))-1) + ((fib2\ (l+1)) - (l+1))$
proof (*induction rl arbitrary*:*u l*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a rl*)
then show *?case*
proof (*cases* $l=2$)
case *True*
then show *?thesis* **using** *Cons.prems Cons.IH*
by (*simp add*: *numeral-2-eq-2 numeral-3-eq-3*)
next
case *False*
from *this* **have** *l3*:$l \ge 3$ **using** *Cons.prems* **by** *simp*
from *this* **have** *a0*: $l - 1 \ge 2$ **by** *simp*
from *Cons.prems* **have** *a1*: $\forall i.\ 3 \le i \wedge i \le l - 1 \longrightarrow$
$(l - 1 - i < l - 1 - 2 \longrightarrow rl\ !\ Suc\ (l - 1 - i) + rl\ !\ Suc\ (Suc\ (l - 1 - i)) < rl\ !\ (l - 1 - i)) \wedge$
$(l - 1 - i < l - 1 - 1 \longrightarrow rl\ !\ Suc\ (l - 1 - i) < rl\ !\ (l - 1 - i)) \wedge u \le rl\ !\ (l - 1 - i) \Longrightarrow$
$rl\ !\ (l - 1 - 1) < rl\ !\ (l - 1 - 2)$

apply (*clarsimp*)
by (*metis False One-nat-def diff-Suc-eq-diff-pred less-than nth-Cons-pos numeral-2-eq-2 zero-less-diff*)
from *Cons.prems a0* **have** *a2*:rl ! $(l - 1 - 1) < rl$! $(l - 1 - 2)$
using *l3* **by** (*simp add*: *numeral-2-eq-2*)
from *Cons.prems* **have** *a3*:$u \leq rl$! $(l - 1 - 1)$
by (*simp*)
from *Cons.prems* **have** *a4*:*length rl* $= l - 1$
by (*simp*)
from *Cons.IH*[*of l−1 u*] *a0 a1 a2 a3 a4* **have** *l1*:$u * (fib\ (l - 1 + 1) - 1) + (fib2\ (l - 1 + 1) - (l - 1 + 1)) \leq sumn\ rl\ (l - 1)$
apply *clarsimp*
using *Cons.prems* **by** (*smt Suc-diff-le Suc-le-lessD Suc-less-eq Suc-pred diff-Suc-1 diff-Suc-Suc nat-less-le nth-Cons-pos numeral-2-eq-2 zero-less-Suc*)
from *Cons.prems* **have** *la*:$a \geq u*(fib\ (l-1)) + (fib2\ (l-1))$ **using** *rl-elem-lower-bound*[*of l a#rl u l−1*]
by (*metis One-nat-def a4 diff-self-eq-0 less-add-same-cancel1 less-numeral-extra*(*1*) *list.size*(*4*) *nth-Cons-0*)
have $fib2\ l + fib2\ (l - Suc\ 0) + 1 = fib2\ (Suc\ l)$ **using** *fib2.simps*(*3*)[*of l − Suc 0*] *Cons.prems*(*5*) **by** *simp*
from *this* **have** $fib2\ l + fib2\ (l - Suc\ 0) + 1 - Suc\ l = fib2\ (Suc\ l) - Suc\ l$ **using** *minus-same-num* **by** *simp*
from *this* **have** $fib2\ l + fib2\ (l - Suc\ 0) - l = fib2\ (Suc\ l) - Suc\ l$ **using** *minus-suc-plus-one* **by** *simp*
from *this* **have** *f2*:$fib2\ l - l + fib2\ (l - Suc\ 0) = fib2\ (Suc\ l) - Suc\ l$ **using** *minus-exc fib2-1* **by** *metis*
have $fib\ l + fib\ (l - Suc\ 0) = fib\ (Suc\ l)$ **using** *fib.simps*(*3*) *Cons.prems*(*5*) **by** (*metis Cons.prems*(*4*) *One-nat-def a4 add.commute length-Cons*)
from *this* **have** $(fib\ l - Suc\ 0) + fib\ (l - Suc\ 0) = (fib\ (Suc\ l) - Suc\ 0)$ **using** *Cons.prems*(*5*) *minus-exc fib-1* **by** *metis*
from *this* **have** *f1*:$u * (fib\ l - Suc\ 0) + u * fib\ (l - Suc\ 0) = u * (fib\ (Suc\ l) - Suc\ 0)$ **by** (*metis add-mult-distrib2*)
from *l1 la* **show** *?thesis*
apply (*subgoal-tac sumn* $(a\ \#\ rl)\ l = a + sumn\ rl\ (l-1)$)
apply *clarsimp*
apply (*subgoal-tac* $u * (fib\ (Suc\ (l - Suc\ 0)) - Suc\ 0) + u * fib\ (l - Suc\ 0) = u * (fib\ (Suc\ l) - Suc\ 0)$
$(fib2\ (Suc\ (l - Suc\ 0)) - Suc\ (l - Suc\ 0)) + fib2\ (l - Suc\ 0) = (fib2\ (Suc\ l) - Suc\ l)$)
apply *simp*
using *Cons.prems*(*5*) *f1 f2* **apply** *simp-all*
using *Cons.prems sumn-first-one-out* **by** *simp*
qed
qed

lemma *l119*[*simp*]: $16 * (fib\ 5 - Suc\ 0) + (fib2\ 5 - 5) = 119$
sorry
lemma *l1541*[*simp*]: $16 * (fib\ 10 - Suc\ 0) + (fib2\ 10 - 10) = 1541$
sorry

```
lemma l119150[simp]: 16 ∗ (fib 19 − Suc 0) + (fib2 19 − 19) = 119150
  sorry
lemma l2917[simp]: 16 ∗ (fib 40 − Suc 0) + (fib2 40 − 40) = 2917196495
  sorry


lemma run-len-elem-lower-bound:
∀ i. 3≤i ∧ i≤l ⟶ elem-inv rl (l−i) u ⟹
elem-bigger-than-next rl (l−2) ⟹
elem-larger-than-bound rl (l−1) u ⟹ length rl = l ⟹ l≥2 ⟹ k<l
⟹ rl!(l−1−k) ≥ u∗(fib k) + (fib2 k)
 apply (simp only:elem-inv-def elem-larger-than-bound-def elem-bigger-than-next-2-def
elem-bigger-than-next-def)
  apply (rule rl-elem-lower-bound)
       apply auto
  by (metis Suc-diff-le diff-Suc-Suc numeral-2-eq-2)

lemma run-len-sum-lower-bound:
∀ i. 3≤i ∧ i≤l ⟶ elem-inv rl (l−i) u ⟹
elem-bigger-than-next rl (l−2) ⟹
elem-larger-than-bound rl (l−1) u ⟹ length rl = l ⟹ l≥2
⟹ sumn rl l ≥ u∗((fib (l+1))−1) + ((fib2 (l+1)) − (l+1))
  apply (rule rl-sum-lower-bound)
    apply (auto simp add:elem-inv-def elem-larger-than-bound-def elem-bigger-than-next-2-def
elem-bigger-than-next-def)
  by (metis Suc-diff-le diff-Suc-Suc numeral-2-eq-2)

lemma
elem-inv rl 0 u ⟹
 elem-inv rl 1 u ⟹
elem-bigger-than-next rl 2 ⟹
elem-larger-than-bound rl 3 u ⟹
length rl = 4 ⟹ u ≥ 16
⟹ sum rl ≥ 119
 apply (simp add:elem-inv-def elem-larger-than-bound-def elem-bigger-than-next-2-def
elem-bigger-than-next-def)
  apply (case-tac rl)
   prefer 2
   apply (case-tac list)
    prefer 2
    apply (case-tac lista)
     prefer 2
     apply (case-tac listb)
      apply auto
  done


end
```

5 The Simpl Syntax

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

5.1 The Core Language

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

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

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

5.2 Derived Language Constructs

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

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

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

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

lemma $fst\text{-}guaranteeStripPair$: $fst\ (guaranteeStripPair\ f\ g) = f$
by $(simp\ add:\ guaranteeStripPair\text{-}def)$

lemma $snd\text{-}guaranteeStripPair$: $snd\ (guaranteeStripPair\ f\ g) = g$
by $(simp\ add:\ guaranteeStripPair\text{-}def)$

5.3 Operations on Simpl-Syntax

5.3.1 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*:: $(('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com) \Rightarrow$
$('s,'p,'f)\ com\ list \Rightarrow ('s,'p,'f)\ com$
where
$sequence\ seq\ []\ =\ Skip\ |$
$sequence\ seq\ (c\#cs) = (case\ cs\ of\ []\ \Rightarrow c$
$|\ \text{-} \Rightarrow seq\ c\ (sequence\ seq\ cs))$

primrec *normalize*:: $('s,'p,'f)\ com \Rightarrow ('s,'p,'f)\ com$
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*: $\forall\ c \in set\ (flatten\ c').\ flatten\ c = [c]$
apply (*induct c'*)
apply *simp*
apply *simp*
apply *simp*
apply (*simp* (*no-asm-use*))
apply *blast*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
apply (*simp* (*no-asm-use*))
apply *simp*
apply (*simp* (*no-asm-use*))
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*
| (*x'*#*xs'*) $\Rightarrow$ *normalize c*= *Seq x* (*sequence Seq xs*))
proof (*induct c*)
case (*Seq c1 c2*)
have *flatten* (*normalize* (*Seq c1 c2*)) = *x* # *xs* **by** *fact*
hence *flatten* (*sequence Seq* (*flatten* (*normalize c1*) @ *flatten* (*normalize c2*))) =
x#*xs*
by *simp*
hence *x-xs*: *flatten* (*normalize c1*) @ *flatten* (*normalize c2*) = *x* # *xs*
by (*simp add*: *flatten-app*)
show *?case*
proof (*cases flatten* (*normalize c1*))
case *Nil*
with *flatten-nonEmpty* **show** *?thesis* **by** *auto*
next
case (*Cons x1 xs1*)
note *Cons-x1-xs1* = *this*
with *x-xs* **obtain**
x-x1: *x*=*x1* **and** *xs-rest*: *xs*=*xs1*@*flatten* (*normalize c2*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.3.2 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* (λ*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* ⇒
 (*'p* ⇒ (*'s*,*'p*,*'f*) *com option*) ⇒ (*'p* ⇒ (*'s*,*'p*,*'f*) *com option*)
 where *strip F* Γ = (λ*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* (λ*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* (λ(*f*,*g*). *f* $\notin$ *F*) *gs*) *b I V* (*strip-guards F c*)
by (*simp add*: *whileAnnoG-def*)

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

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

5.3.3 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 (λ*s t*. *mark-guards f* (*c s t*))
by (*simp add*: *dynCall-def*)

lemma *mark-guards-fcall* [*simp*]:
mark-guards f (*fcall init p return result c*) =
fcall init p return result (λ*v*. *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 (λ*s*. *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*

5.3.4 Merging Guards: *merge-guards*

primrec *merge-guards*:: ('s,'p,'f) *com* ⇒ ('s,'p,'f) *com*
where
merge-guards Skip = *Skip* |
merge-guards (*Basic g*) = *Basic g* |

merge-guards (*Spec* *r*) = *Spec* *r* |
merge-guards (*Seq* c_1 c_2) = (*Seq* (*merge-guards* c_1) (*merge-guards* c_2)) |
merge-guards (*Cond* *b* c_1 c_2) = *Cond* *b* (*merge-guards* c_1) (*merge-guards* c_2) |
merge-guards (*While* *b* *c*) = *While* *b* (*merge-guards* *c*) |
merge-guards (*Call* *p*) = *Call* *p* |
merge-guards (*DynCom* *c*) = *DynCom* (λs. (*merge-guards* (*c* *s*))) |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *merge-guards-bind* [*simp*]:
merge-guards (*bind e c*) = *bind e* (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 ($\lambda s\ t$. *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 ($\lambda s\ t$. *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 (λv. *merge-guards* (*c v*))

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

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

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

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

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

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

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

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

5.3.5 Intersecting Guards: $c_1 \cap_g c_2$

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

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

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

apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*, *simp*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*)
apply (*erule allE*, *erule mp*, (*rule allI impI*)+, *erule com-rel-elim-cases*,*simp*,*simp*)
done

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

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

recdef *inter-guards inv-image com-rel fst*
(*Skip* $\cap_g$ *Skip*) = *Some Skip*
(*Basic f1* $\cap_g$ *Basic f2*) = (*if f1* = *f2 then Some* (*Basic f1*) *else None*)
(*Spec r1* $\cap_g$ *Spec r2*) = (*if r1* = *r2 then Some* (*Spec r1*) *else None*)
(*Seq a1 a2* $\cap_g$ *Seq b1 b2*) =
  (*case a1* $\cap_g$ *b1 of*
    None $\Rightarrow$ *None*
  | *Some c1* $\Rightarrow$ (*case a2* $\cap_g$ *b2 of*
      None $\Rightarrow$ *None*
    | *Some c2* $\Rightarrow$ *Some* (*Seq c1 c2*)))
(*Cond cnd1 t1 e1* $\cap_g$ *Cond cnd2 t2 e2*) =
  (*if cnd1* = *cnd2*
   then (*case t1* $\cap_g$ *t2 of*
      None $\Rightarrow$ *None*
    | *Some t* $\Rightarrow$ (*case e1* $\cap_g$ *e2 of*
        None $\Rightarrow$ *None*
      | *Some e* $\Rightarrow$ *Some* (*Cond cnd1 t e*)))
   else None)
(*While cnd1 c1* $\cap_g$ *While cnd2 c2*) =
  (*if cnd1* = *cnd2*
   then (*case c1* $\cap_g$ *c2 of*
      None $\Rightarrow$ *None*
    | *Some c* $\Rightarrow$ *Some* (*While cnd1 c*))
   else None)
(*Call p1* $\cap_g$ *Call p2*) =
  (*if p1* = *p2*
   then Some (*Call p1*)
   else None)
(*DynCom P1* $\cap_g$ *DynCom P2*) =
  (*if* ($\forall s.$ (*P1 s* $\cap_g$ *P2 s*) $\neq$ *None*)

then Some (*DynCom* (λ*s*. *the* (*P1 s* $\cap_g$ *P2 s*)))
else None)
(*Guard m1 g1 c1* $\cap_g$ *Guard m2 g2 c2*) =
(*if m1* = *m2 then*
(*case c1* $\cap_g$ *c2 of*
None $\Rightarrow$ *None*
| *Some c* $\Rightarrow$ *Some* (*Guard m1* (*g1* $\cap$ *g2*) *c*))
else None)
(*Throw* $\cap_g$ *Throw*) = *Some Throw*
(*Catch a1 a2* $\cap_g$ *Catch b1 b2*) =
(*case a1* $\cap_g$ *b1 of*
None $\Rightarrow$ *None*
| *Some c1* $\Rightarrow$ (*case a2* $\cap_g$ *b2 of*
None $\Rightarrow$ *None*
| *Some c2* $\Rightarrow$ *Some* (*Catch c1 c2*)))
(*c* $\cap_g$ *d*) = *None*
(**hints** *cong add*: *option.case-cong if-cong*
recdef-wf: *wf-com-rel simp*: *com-rel.intros*)

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

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

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

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

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

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

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

lemma *inter-guards-Call*:
$(Call\ p \cap_g c2) = Some\ c =$
$(c2{=}Call\ p \wedge c{=}Call\ p)$
by (*cases c2*) (*auto split*: *if-split-asm*)

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

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

lemma *inter-guards-Throw*:

$(\mathit{Throw} \cap_g c2) = \mathit{Some}\ c = (c2{=}\mathit{Throw} \wedge c{=}\mathit{Throw})$
by (*cases c2*) *auto*

lemma *inter-guards-Catch*:
$(\mathit{Catch}\ a1\ a2 \cap_g c2) = \mathit{Some}\ c =$
$(\exists b1\ b2\ d1\ d2.\ c2{=}\mathit{Catch}\ b1\ b2 \wedge (a1 \cap_g b1) = \mathit{Some}\ d1 \wedge$
$(a2 \cap_g b2) = \mathit{Some}\ d2 \wedge c{=}\mathit{Catch}\ d1\ d2)$
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*

5.3.6 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**
$\mathit{Skip} \subseteq_g \mathit{Skip}$
$\mid f1 = f2 \Longrightarrow \mathit{Basic}\ f1 \subseteq_g \mathit{Basic}\ f2$
$\mid r1 = r2 \Longrightarrow \mathit{Spec}\ r1 \subseteq_g \mathit{Spec}\ r2$
$\mid a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow \mathit{Seq}\ a1\ a2 \subseteq_g \mathit{Seq}\ b1\ b2$
$\mid cnd1 = cnd2 \Longrightarrow t1 \subseteq_g t2 \Longrightarrow e1 \subseteq_g e2 \Longrightarrow \mathit{Cond}\ cnd1\ t1\ e1 \subseteq_g \mathit{Cond}\ cnd2$
$t2\ e2$
$\mid cnd1 = cnd2 \Longrightarrow c1 \subseteq_g c2 \Longrightarrow \mathit{While}\ cnd1\ c1 \subseteq_g \mathit{While}\ cnd2\ c2$
$\mid p1 = p2 \Longrightarrow \mathit{Call}\ p1 \subseteq_g \mathit{Call}\ p2$
$\mid (\bigwedge s.\ P1\ s \subseteq_g P2\ s) \Longrightarrow \mathit{DynCom}\ P1 \subseteq_g \mathit{DynCom}\ P2$
$\mid m1 = m2 \Longrightarrow g1 = g2 \Longrightarrow c1 \subseteq_g c2 \Longrightarrow \mathit{Guard}\ m1\ g1\ c1 \subseteq_g \mathit{Guard}\ m2\ g2\ c2$
$\mid c1 \subseteq_g c2 \Longrightarrow c1 \subseteq_g \mathit{Guard}\ m2\ g2\ c2$
$\mid \mathit{Throw} \subseteq_g \mathit{Throw}$
$\mid a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow \mathit{Catch}\ a1\ a2 \subseteq_g \mathit{Catch}\ b1\ b2$

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

lemma *subseteq-guards-Basic*:
$c = \mathit{Basic}\ f$ **if** $c \subseteq_g \mathit{Basic}\ f$
using *that* **by** *cases simp*

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

lemma *subseteq-guards-Seq*:
$\exists c1'\ c2'.\ c = \mathit{Seq}\ c1'\ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$ **if** $c \subseteq_g \mathit{Seq}\ c1\ c2$
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

6 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*; ⋀*s*. *S*=*Abrupt s* ⟹ *P*⟧ ⟹ *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*; ⋀*f*. *s*=*Fault f* ⟹ *P*⟧ ⟹ *P*
  by (*auto simp add*: *isFault-def*)

lemma *not-isFault-iff*: (¬ *isFault t*) = (∀ *f*. *t* ≠ *Fault f*)
  by (*auto elim*: *isFaultE*)

6.1 Big-Step Execution: Γ⊢⟨*c*, *s*⟩ ⇒ *t*

The procedure environment

type-synonym ('*s*,'*p*,'*f*) *body* = '*p* ⇒ ('*s*,'*p*,'*f*) *com option*

inductive
  exec::[('*s*,'*p*,'*f*) *body*,('*s*,'*p*,'*f*) *com*,('*s*,'*f*) *xstate*,('*s*,'*f*) *xstate*]
                    ⇒ *bool* (-⊢ ⟨-,-⟩ ⇒ - [*60*,*20*,*98*,*98*] *89*)
  for Γ::('*s*,'*p*,'*f*) *body*
where
  Skip: Γ⊢⟨*Skip*,*Normal s*⟩ ⇒ *Normal s*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *exec-block*:

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

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

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

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

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

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

lemma *exec-callFault*:

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

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

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

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

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

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

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

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

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

apply *simp*
apply (*case-tac s'*)
apply *simp-all*
apply (*elim exec-Normal-elim-cases*)
apply *simp*
apply (*rule Normal*, *assumption+*)
apply (*drule Fault-end*) **apply** *simp*
apply (*rule Fault*,*assumption+*)
apply (*drule Stuck-end*) **apply** *simp*
apply (*rule Stuck*,*assumption+*)
done

lemma *exec-call-Normal-elim* [*consumes 1*]:
assumes *exec-call*: $\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle \Rightarrow\ t$
assumes *Normal*:
$\bigwedge bdy\ t'.$
$[\![\Gamma\ p = Some\ bdy;\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow\ Normal\ t';$
$\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle \Rightarrow\ t]\!]$
$\Longrightarrow P$
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*:
$[\![\Gamma\vdash\langle \textit{call init } (p\ s) \textit{ return } c,\textit{Normal } s\rangle \Rightarrow\ t]\!]$
$\Longrightarrow$
$\Gamma\vdash\langle \textit{dynCall init } p \textit{ return } c,\textit{Normal } s\rangle \Rightarrow\ t$
apply (*simp add*: *dynCall-def*)
by (*rule DynCom*)

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

lemma *exec-Call-body*:
$\Gamma\ p=\textit{Some bdy} \Longrightarrow$
$\Gamma\vdash\langle \textit{Call } p,s\rangle \Rightarrow\ t = \Gamma\vdash\langle \textit{the } (\Gamma\ p),s\rangle \Rightarrow\ t$
apply (*rule*)
apply (*fastforce elim*: *exec-elim-cases*)

apply (*cases s*)
apply (*cases t*)
apply (*fastforce intro*: *exec.intros dest*: *Fault-end Abrupt-end Stuck-end*)+
done

lemma *exec-Seq'*: $\llbracket\Gamma\vdash\langle c1,s\rangle \Rightarrow\ s'$; $\Gamma\vdash\langle c2,s'\rangle \Rightarrow\ s''\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle Seq\ c1\ c2,s\rangle \Rightarrow\ s''$
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'*)

6.2 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*: $\llbracket s\in g$; $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ t\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle Guard\ f\ g\ c,Normal\ s\rangle =n\Rightarrow\ t$

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

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

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

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

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

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

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

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

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

| *WhileFalse*: $\llbracket s \notin b\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ s\rangle =n\Rightarrow\ Normal\ s$

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

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

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

| *DynCom*: $\llbracket\Gamma\vdash\langle(c\ s),Normal\ s\rangle =n\Rightarrow\ t\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle DynCom\ c,Normal\ s\rangle =n\Rightarrow\ t$

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

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

| *CatchMatch*: $\llbracket\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow\ Abrupt\ s';\ \Gamma\vdash\langle c_2,Normal\ s'\rangle =n\Rightarrow\ t\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$
| *CatchMiss*: $\llbracket\Gamma\vdash\langle c_1,Normal\ s\rangle =n\Rightarrow\ t;\ \neg isAbr\ t\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle =n\Rightarrow\ t$

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

lemma *execn-callUndefined*:
$\llbracket\Gamma\ p{=}None\rrbracket$
$\Longrightarrow$
$\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle\ {=}Suc\ n{\Rightarrow}\ \ Stuck$
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*:
$\bigwedge t'.$
$\llbracket\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ {=}n{\Rightarrow}\ \ Normal\ t';$
$\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle\ {=}n{\Rightarrow}\ \ t\rrbracket$
$\Longrightarrow P$
assumes *Abrupt*:
$\bigwedge t'.$
$\llbracket\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ {=}n{\Rightarrow}\ \ Abrupt\ t';$
$t = Abrupt\ (return\ s\ t')\rrbracket$
$\Longrightarrow P$
assumes *Fault*:
$\bigwedge f.$
$\llbracket\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ {=}n{\Rightarrow}\ \ Fault\ f;$
$t = Fault\ f\rrbracket$
$\Longrightarrow P$
assumes *Stuck*:
$\llbracket\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ {=}n{\Rightarrow}\ \ Stuck;$
$t = Stuck\rrbracket$
$\Longrightarrow P$
assumes *Undef*:
$\llbracket\Gamma\ p = None;\ t = Stuck\rrbracket \Longrightarrow P$
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 $\bigwedge m.\ n \le m \Longrightarrow \Gamma\vdash\langle c,s\rangle =m\Rightarrow\ t$
using *exec*
by (*induct*) (*auto intro*: *execn.intros dest*: *Suc-le-D*)

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

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

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

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

shows $\exists\, n.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow\ t$
using *execn*
proof (*induct*)
case *Skip* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Guard* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *GuardFault* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *FaultProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Basic* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Spec* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *SpecStuck* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*Seq c1 s s′ c2 s′′*)
then obtain *n m* **where**
$\Gamma\vdash\langle c1,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash\langle c2,s'\rangle =m\Rightarrow\ s''$
by *blast*
then have
$\Gamma\vdash\langle c1,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$
$\Gamma\vdash\langle c2,s'\rangle =max\ n\ m\Rightarrow\ s''$
by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
thus *?case*
by (*iprover intro*: *execn.intros*)
next
case *CondTrue* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *CondFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*WhileTrue s b c s′ s′′*)
then obtain *n m* **where**
$\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow\ s'$ $\Gamma\vdash\langle While\ b\ c,s'\rangle =m\Rightarrow\ s''$
by *blast*
then have
$\Gamma\vdash\langle c,Normal\ s\rangle =max\ n\ m\Rightarrow\ s'$ $\Gamma\vdash\langle While\ b\ c,s'\rangle =max\ n\ m\Rightarrow\ s''$
by (*auto elim*!: *execn-mono intro*: *max.cobounded1 max.cobounded2*)
with *WhileTrue*
show *?case*
by (*iprover intro*: *execn.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Call* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *CallUndefined* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next

case *StuckProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *DynCom* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *Throw* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case *AbruptProp* **thus** *?case* **by** (*iprover intro*: *execn.intros*)
next
case (*CatchMatch c1 s s′ c2 s′′*)
then obtain *n m* **where**
$\Gamma\vdash\langle c1,Normal\ s\rangle\ =n\Rightarrow\ \ Abrupt\ s'\ \Gamma\vdash\langle c2,Normal\ s'\rangle\ =m\Rightarrow\ \ s''$
by *blast*
then have
$\Gamma\vdash\langle c1,Normal\ s\rangle\ =max\ n\ m\Rightarrow\ \ Abrupt\ s'$
$\Gamma\vdash\langle c2,Normal\ s'\rangle\ =max\ n\ m\Rightarrow\ \ s''$
by (*auto elim*!: *execn-mono intro*: *max.coboundedl 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*: $[\![\bigwedge t.\ \Gamma\vdash\langle c,s\rangle \Rightarrow t \Longrightarrow t\neq$*Fault f*$]\!] \Longrightarrow \Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{$*Fault f*$\}$
  by (*simp add*: *final-notin-def*)

lemma *noFaultI'*:

assumes *contr*: $\Gamma\vdash\langle c,s\rangle \Rightarrow$ *Fault f* $\Longrightarrow$ *False*
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow\notin\{$*Fault f*$\}$
proof (*rule noFaultI*)
fix *t* **assume** $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
with *contr* **show** $t \neq$ *Fault f*
by (*cases t=Fault f*) *auto*
qed

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

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

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

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

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

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

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

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

lemma *noFaultn-exec-startD*: $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{$*Fault f*$\}$; $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow s\neq$*Fault*

f
  by (*auto simp add*: *nfinal-notin-def dest*: *noFaultn-startD*)

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

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

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

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

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

lemma *noFaultStuckn-execD*:
  $[\![\Gamma\vdash\langle c,s\rangle =n\Rightarrow\notin\{Fault\ True,Fault\ False,Stuck\};\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t]\!] \Longrightarrow$
      $t\notin\{Fault\ True,Fault\ False,Stuck\}$
  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'*)

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

with *exec-xs-ys* **have** $t=Stuck$
  by (*auto dest*: *execn-Stuck-end*)
with *Stuck exec-x Cons*
show *?thesis*
  by (*auto intro*: *execn.intros*)
qed
qed
qed

lemma *execn-sequence-appE* [*consumes 1*]:
$[\![\Gamma\vdash\langle sequence\ Seq\ (xs\ @\ ys),Normal\ s\rangle =n\Rightarrow t;$
$\bigwedge s'.\ [\![\Gamma\vdash\langle sequence\ Seq\ xs,Normal\ s\rangle =n\Rightarrow s';\Gamma\vdash\langle sequence\ Seq\ ys,s'\rangle =n\Rightarrow t]\!]$
$\Longrightarrow P$
$]\!] \Longrightarrow P$
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*: $\Gamma\vdash\langle sequence\ Seq\ (flatten\ (Seq\ c1\ c2)),s\rangle =n\Rightarrow t$ **by** *fact*
show *?case*
proof (*cases s*)
case (*Normal s′*)
with *exec-seq* **obtain** *s″* **where**
$\Gamma\vdash\langle sequence\ Seq\ (flatten\ c1),Normal\ s'\rangle =n\Rightarrow s''$ **and**
$\Gamma\vdash\langle sequence\ Seq\ (flatten\ c2),s''\rangle =n\Rightarrow t$
by (*auto elim*: *execn-sequence-appE*)
with *Seq.hyps Normal*

show *?thesis*
by (*fastforce intro*: *execn.intros*)
next
case *Abrupt*
with *exec-seq*
show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Abrupt-end*)
next
case *Fault*
with *exec-seq*
show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Fault-end*)
next
case *Stuck*
with *exec-seq*
show *?thesis* **by** (*auto intro*: *execn.intros dest*: *execn-Stuck-end*)
qed
qed *auto*

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

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

next
  case *Guard* **thus** *?case* **by** (*auto intro*: *execn.intros elim*!: *execn-elim-cases*)
next
  case *Throw* **thus** *?case* **by** *simp*
next
  case *Catch* **thus** *?case* **by** (*fastforce intro*: *execn.intros elim*!: *execn-elim-cases*)
qed

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

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

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

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

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

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

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

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

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

6.4 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*: $\Gamma\vdash\langle$*Seq c1' c2'*,*s*$\rangle$ =*n*$\Rightarrow$ *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**
$\Gamma\vdash\langle$*c2'*,*w*$\rangle$ =*n*$\Rightarrow$ *t'* **and**
isFault t $\longrightarrow$ *isFault t'* **and**
$\neg$ *isFault t'* $\longrightarrow$ *t'=t*
by *blast*
with *Normal exec-c1' w'*
show *?thesis*
by (*fastforce intro*: *execn.intros*)
qed
qed
qed
next
case (*Cond b c1' c2'*)
have *exec*: $\Gamma\vdash\langle$*c*,*s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
have *c* $\subseteq_g$ *Cond b c1' c2'* **by** *fact*
from *subseteq-guards-Cond* [*OF this*]
obtain *c1 c2* **where**
c: *c* = *Cond b c1 c2* **and**

c1-c1': *c1* $\subseteq_g$ *c1'* **and**
c2-c2': *c2* $\subseteq_g$ *c2'*
by *blast*
show *?case*
proof (*cases s*)
case (*Fault f*)
with *exec* **have** *t=Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec* **have** *t=Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt s'*)
with *exec* **have** *t=Abrupt s'*
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal s'*)
from *exec* [*simplified c Normal*]
show *?thesis*
proof (*cases*)
assume *s'-in-b*: $s' \in b$
assume $\Gamma\vdash\langle c1, Normal\ s'\rangle =n\Rightarrow t$
with *c1-c1' Normal Cond.hyps* **obtain** *t'* **where**
$\Gamma\vdash\langle c1', Normal\ s'\rangle =n\Rightarrow t'$
$isFault\ t \longrightarrow isFault\ t'$
$\neg\ isFault\ t' \longrightarrow t' = t$
by *blast*
with *s'-in-b Normal* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
next
assume *s'-notin-b*: $s' \notin b$
assume $\Gamma\vdash\langle c2, Normal\ s'\rangle =n\Rightarrow t$
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: Γ⊢⟨c,s⟩ =n⇒ t by fact
  have c ⊆g Guard f' g' c' by fact
  hence subset-cases: (c ⊆g c') ∨ (∃ c''. c = Guard f' g' c'' ∧ (c'' ⊆g c'))
    by (rule subseteq-guards-Guard)
  show ?case
```

proof (*cases s*)
case (*Fault f*)
with *exec* **have** t=*Fault f*
by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec* **have** t=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt* s')
with *exec* **have** t=*Abrupt* s'
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s')
from *subset-cases* **show** *?thesis*
proof
assume c-c': $c \subseteq_g c'$
from *Guard.hyps* [*OF this exec*] *Normal* **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c',$*Normal* $s'\rangle$ =n⇒ t' **and**
t-Fault: *isFault* $t \longrightarrow$ *isFault* t' **and**
t-noFault: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
with *Normal*
show *?thesis*
by (*cases* $s' \in g'$) (*fastforce intro*: *execn.intros*)+
next
assume $\exists\, c''.\ c =$ *Guard* $f'\ g'\ c'' \wedge (c'' \subseteq_g c')$
then obtain c'' **where**
c: $c =$ *Guard* $f'\ g'\ c''$ **and**
c''-c': $c'' \subseteq_g c'$
by *blast*
from *c exec Normal*
have *exec-Guard*$'$: $\Gamma\vdash\langle$*Guard* $f'\ g'\ c''$,*Normal* $s'\rangle$ =n⇒ 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⇒ t
from *Guard.hyps* [*OF* c''-c' *exec*-c''] **obtain** t' **where**
exec-c': $\Gamma\vdash\langle c'$,*Normal* $s'\rangle$ =n⇒ 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' ∉ g' t=Fault f'
      with Normal show ?thesis
        by (fastforce intro: execn.intros)
    qed
  qed
qed
next
  case Throw thus ?case
    by (fastforce dest: subseteq-guardsD intro: execn.intros
         elim: execn-elim-cases)
next
  case (Catch c1' c2')
  have c ⊆g Catch c1' c2' by fact
  from subseteq-guards-Catch [OF this]
  obtain c1 c2 where
    c: c = Catch c1 c2 and
    c1-c1': c1 ⊆g c1' and
    c2-c2': c2 ⊆g c2'
    by blast
  have exec: Γ⊢⟨c,s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec have t=Fault f
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
    with exec have t=Stuck
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
    case (Abrupt s')
    with exec have t=Abrupt s'
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
    case (Normal s')
    from exec [simplified c Normal]
    show ?thesis
    proof (cases)
      fix w
```

```
      assume exec-c1: Γ⊢⟨c1,Normal s'⟩ =n⇒ Abrupt w
      assume exec-c2: Γ⊢⟨c2,Normal w⟩ =n⇒ t
      from Normal exec-c1 c1-c1' Catch.hyps obtain w' where
        exec-c1': Γ⊢⟨c1',Normal s'⟩ =n⇒ w' and
        w'-noFault: ¬ isFault w' ⟶ w' = Abrupt w
        by blast
      show ?thesis
      proof (cases isFault w')
        case True
        with exec-c1' Normal show ?thesis
          by (fastforce intro: execn.intros elim: isFaultE)
      next
        case False
        with w'-noFault have w': w'=Abrupt w by simp
        from Normal exec-c2 c2-c2' Catch.hyps obtain t' where
          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 ⊆g c'
  assumes  exec: Γ⊢⟨c,s⟩ ⇒ t
```

shows $\exists t'. \Gamma\vdash\langle c',s\rangle \Rightarrow t' \wedge$
$(\mathit{isFault}\ t \longrightarrow \mathit{isFault}\ t') \wedge (\neg\ \mathit{isFault}\ t' \longrightarrow t'=t)$
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

6.5 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* $\exists f'\ g'\ c'.\ \mathit{merge\text{-}guards}\ c = \mathit{Guard}\ f'\ g'\ c'$)
case *False*
with *s-notin-g*

    show *?thesis*
      by (*cases merge-guards c*) (*auto intro*: *execn.intros simp add*: *Let-def*)
  next
    case *True*
    then obtain f' g' c' **where**
      merge-guards-c: *merge-guards c* = *Guard* f' g' c'
      by *iprover*
    show *?thesis*
    proof (*cases* $f{=}f'$)
      case *False*
      from *s-notin-g merge-guards-c False* **show** *?thesis*
        by (*auto intro*: *execn.intros simp add*: *Let-def*)
    next
      case *True*
      from *s-notin-g merge-guards-c True* **show** *?thesis*
        by (*fastforce intro*: *execn.intros*)
    qed
  qed
qed (*fastforce intro*: *execn.intros*)+

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

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

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

show *?thesis*
by (*auto intro*: *execn.intros*)
next
case *True*
note *f-eq-f′* = *this*
with *merge-guards-c* **have**
merge-guards-Guard: *merge-guards* (*Guard f g c*) = *Guard f* ($g \cap g'$) 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 $\Gamma\vdash\langle$*merge-guards c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *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 $\Gamma\vdash\langle$*merge-guards c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *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 $\Gamma\vdash\langle$*merge-guards* (*Catch c1 c2*),*Normal s*$\rangle$ =*n*$\Rightarrow$ *t* **by** *fact*
hence $\Gamma\vdash\langle$*Catch* (*merge-guards c1*) (*merge-guards c2*),*Normal s*$\rangle$ =*n*$\Rightarrow$ *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*:
$\Gamma\vdash\langle c,\ s\rangle =n\Rightarrow\ t = \Gamma\vdash\langle \mathit{merge\text{-}guards}\ c{,}s\rangle =n\Rightarrow\ t$
by (*blast intro*: *execn-merge-guards-to-execn execn-to-execn-merge-guards*)

theorem *exec-iff-exec-merge-guards*:
$\Gamma\vdash\langle c,\ s\rangle \Rightarrow\ t = \Gamma\vdash\langle \mathit{merge\text{-}guards}\ c{,}s\rangle \Rightarrow\ t$
by (*blast dest*: *exec-to-execn intro*: *execn-to-exec*
intro: *execn-to-execn-merge-guards*
execn-merge-guards-to-execn)

corollary *exec-to-exec-merge-guards*:
$\Gamma\vdash\langle c,\ s\rangle \Rightarrow\ t \Longrightarrow \Gamma\vdash\langle \mathit{merge\text{-}guards}\ c{,}s\rangle \Rightarrow\ t$
by (*rule iffD1* [*OF exec-iff-exec-merge-guards*])

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

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

lemma *execn-to-execn-mark-guards-Fault*:
assumes *exec-c*: $\Gamma\vdash\langle c{,}s\rangle =n\Rightarrow\ t$
shows $\bigwedge f.\ [\![t{=}\mathit{Fault}\ f]\!] \Longrightarrow \exists f'.\ \Gamma\vdash\langle \mathit{mark\text{-}guards}\ x\ c{,}s\rangle =n\Rightarrow\ \mathit{Fault}\ f'$
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: Γ⊢⟨c,Normal s⟩ =n⇒ w by fact
```

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

next
  case *DynCom* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *Throw* **thus** *?case* **by** *simp*
next
  case *AbruptProp* **thus** *?case* **by** *simp*
next
  case (*CatchMatch c1 s n w c2 t*)
  have *exec-c1*: $\Gamma\vdash\langle c1, Normal\ s\rangle =n\Rightarrow Abrupt\ w$ **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*: $\Gamma\vdash\langle mark\text{-}guards\ f\ (Seq\ c1\ c2), s\rangle =n\Rightarrow t$ **by** *fact*
  then obtain *w* **where**
    exec-mark-c1: $\Gamma\vdash\langle mark\text{-}guards\ f\ c1, s\rangle =n\Rightarrow w$ **and**
    exec-mark-c2: $\Gamma\vdash\langle mark\text{-}guards\ f\ c2, w\rangle =n\Rightarrow t$
    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: ¬ isFault w' ⟶ 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'∈ b)
      case True
      with Normal exec-mark
```

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

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

```
      r′-noFault: ¬ isFault r′ ⟶ r′=r
      by blast
    show ?thesis
    proof (cases isFault r′)
      case False
      with r′-noFault have r′: r′=r by simp
      from hyp-while exec-mark-w
      obtain t′ where
        Γ⊢⟨While b c,r⟩ =n⇒ t′
        isFault t ⟶ isFault t′
        t′ = Fault f ⟶ t′=t
        isFault t′ ⟶ isFault t
        ¬ isFault t′ ⟶ t′=t
        by blast
      with r′ exec-c Normal s′-in-b
      show ?thesis
        by (blast intro: execn.intros)
    next
      case True
      then obtain f′ where r′: r′=Fault f′..
      hence Γ⊢⟨While b c,r′⟩ =n⇒ Fault f′
        by auto
      with Normal s′-in-b exec-c
      have exec: Γ⊢⟨While b c,Normal s′⟩ =n⇒ Fault f′
        by (auto intro: execn.intros)
      from True r′-Fault
      have isFault r
        by simp
      then obtain f where r: r=Fault f..
      with exec-mark-w have t=Fault f
        by (auto dest: execn-Fault-end)
      with Normal exec r′ r r′-Fault-f
      show ?thesis
        by auto
    qed
  qed
qed
next
  case Call thus ?case by auto
next
  case DynCom thus ?case
    by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
  case (Guard f′ g c s n t)
  have exec-mark: Γ⊢⟨mark-guards f (Guard f′ g c),s⟩ =n⇒ t by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-mark have t=Fault f
```

by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
by *auto*
next
case *Stuck*
with *exec-mark* **have** t=*Stuck*
by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
by *auto*
next
case (*Abrupt* s')
with *exec-mark* **have** t=*Abrupt* s'
by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
by *auto*
next
case (*Normal* s')
show *?thesis*
proof (*cases* $s' \in g$)
case *False*
with *Normal exec-mark* **have** t: t=*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⇒ t **by** *simp*
  thus *?thesis*
  proof (*cases*)
    fix *w*
    assume *exec-mark-c1*: $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s'*$\rangle$ =n⇒ *Abrupt w*
    assume *exec-mark-c2*: $\Gamma\vdash\langle$*mark-guards f c2*,*Normal w*$\rangle$ =n⇒ t
    from *exec-mark-c1 Catch.hyps*
    obtain *w'* **where**
      exec-c1: $\Gamma\vdash\langle$*c1*,*Normal s'*$\rangle$ =n⇒ *w'* **and**
      w'-Fault-f: *w'* = *Fault f* ⟶ *w'*=*Abrupt w* **and**
      w'-Fault: *isFault w'* ⟶ *isFault* (*Abrupt w*) **and**
      w'-noFault: ¬ *isFault w'* ⟶ *w'*=*Abrupt w*
      by *fastforce*
    show *?thesis*
    proof (*cases w'*)
      case (*Fault f'*)
      with *Normal exec-c1* **have** $\Gamma\vdash\langle$*Catch c1 c2*,*s*$\rangle$ =n⇒ *Fault f'*
        by (*auto intro*: *execn.intros*)
      with *w'-Fault Fault* **show** *?thesis*
        by *auto*
    next
      case *Stuck*
      with *w'-noFault* **have** *False*
        by *simp*
      thus *?thesis* **..**
    next
      case (*Normal w''*)

with *w′-noFault* **have** *False* **by** *simp* **thus** *?thesis* **..**
next
case (*Abrupt w′′*)
with *w′-noFault* **have** *w′′*: *w′′=w* **by** *simp*
from *exec-mark-c2 Catch.hyps*
obtain *t′* **where**
$\Gamma\vdash\langle$*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 $\exists f'.\ \Gamma\vdash\langle mark\text{-}guards\ x\ c,s\rangle \Rightarrow Fault\ f'$
proof –
  from *exec-to-execn* [*OF exec-c*] **obtain** *n* **where**
    $\Gamma\vdash\langle c,s\rangle =n\Rightarrow Fault\ f$ **..**
  from *execn-to-execn-mark-guards-Fault* [*OF this*]
  show *?thesis*
    by (*blast intro*: *execn-to-exec*)
qed

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

6.7 Lemmas about *strip-guards*

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

lemma *execn-to-execn-strip-guards-Fault*:
 assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
 shows $\bigwedge f.\ [\![t=Fault\ f;\ f \notin F]\!] \Longrightarrow \Gamma\vdash\langle strip\text{-}guards\ F\ c,s\rangle =n\Rightarrow Fault\ f$
using *exec-c*

proof (*induct*)
  case *Skip* **thus** *?case* **by** *auto*
next
  case *Guard* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *GuardFault* **thus** *?case* **by** (*fastforce intro*: *execn.intros*)
next
  case *FaultProp* **thus** *?case* **by** *auto*
next
 case *Basic* **thus** *?case* **by** *auto*
next
 case *Spec* **thus** *?case* **by** *auto*
next
 case *SpecStuck* **thus** *?case* **by** *auto*
next
  case (*Seq c1 s n w c2 t*)
  have *exec-c1*: Γ⊢⟨*c1*,*Normal s*⟩ =*n*⇒ *w* **by** *fact*
  have *exec-c2*: Γ⊢⟨*c2*,*w*⟩ =*n*⇒ *t* **by** *fact*
  have *t*: *t*=*Fault f* **by** *fact*
  have *notinF*: *f* ∉ *F* **by** *fact*
  show *?case*
  proof (*cases w*)
    case (*Fault f'*)
    with *exec-c2 t* **have** *f'*=*f*
      by (*auto dest*: *execn-Fault-end*)
    with *Fault notinF Seq.hyps*
    have Γ⊢⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *Fault f*
      by *auto*
    moreover have Γ⊢⟨*strip-guards F c2*,*Fault f*⟩ =*n*⇒ *Fault f*
      by *auto*
    ultimately show *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case (*Normal s'*)
    with *execn-to-execn-strip-guards* [*OF exec-c1*]
    have *exec-strip-c1*: Γ⊢⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *w*
      by *simp*
    with *Seq.hyps t notinF*
    have Γ⊢⟨*strip-guards F c2*,*w*⟩ =*n*⇒ *Fault f*
      by *blast*
    with *exec-strip-c1* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case (*Abrupt s'*)
    with *execn-to-execn-strip-guards* [*OF exec-c1*]
    have *exec-strip-c1*: Γ⊢⟨*strip-guards F c1*,*Normal s*⟩ =*n*⇒ *w*
      by *simp*
    with *Seq.hyps t notinF*
    have Γ⊢⟨*strip-guards F c2*,*w*⟩ =*n*⇒ *Fault f*

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

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

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

qed (*insert exec-c*, *auto intro*: *execn-to-execn-strip-guards*)

lemma *execn-strip-guards-to-execn*:
  $\bigwedge s\ n\ t.\ \Gamma\vdash\langle \textit{strip-guards}\ F\ c,s\rangle =n\Rightarrow t$
  $\Longrightarrow \exists\, t'.\ \Gamma\vdash\langle c,s\rangle =n\Rightarrow t' \wedge$
          $(\textit{isFault}\ t \longrightarrow \textit{isFault}\ t') \wedge$
          $(t' \in \textit{Fault}\ `\ (-\ F) \longrightarrow t'{=}t) \wedge$
          $(\neg\ \textit{isFault}\ t' \longrightarrow t'{=}t)$
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*: $\Gamma\vdash\langle \textit{strip-guards}\ F\ (\textit{Seq}\ c1\ c2),s\rangle =n\Rightarrow t$ **by** *fact*
  then obtain w **where**
    exec-strip-c1: $\Gamma\vdash\langle \textit{strip-guards}\ F\ c1,s\rangle =n\Rightarrow w$ **and**
    exec-strip-c2: $\Gamma\vdash\langle \textit{strip-guards}\ F\ c2,w\rangle =n\Rightarrow t$
    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: $\textit{isFault}\ w \longrightarrow \textit{isFault}\ w'$ **and**
    w'-Fault: $w' \in \textit{Fault}\ `\ (-\ F) \longrightarrow w'{=}w$ **and**
    w'-noFault: $\neg\ \textit{isFault}\ w' \longrightarrow w'{=}w$
    by *blast*
  show *?case*
  proof (*cases s*)
    case (*Fault f*)
    with *exec-strip* **have** $t{=}\textit{Fault}\ f$
      by (*auto dest*: *execn-Fault-end*)
    with *Fault* **show** *?thesis*
      by *auto*
  next
    case *Stuck*
    with *exec-strip* **have** $t{=}\textit{Stuck}$
      by (*auto dest*: *execn-Stuck-end*)
    with *Stuck* **show** *?thesis*
      by *auto*
  next
    case (*Abrupt s'*)
    with *exec-strip* **have** $t{=}\textit{Abrupt}\ s'$
      by (*auto dest*: *execn-Abrupt-end*)
    with *Abrupt* **show** *?thesis*
      by *auto*
  next
    case (*Normal s'*)

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

with *exec-strip* **have** t=*Fault f*
  by (*auto dest*: *execn-Fault-end*)
with *Fault* **show** *?thesis*
  by *auto*

next
case *Stuck*
with *exec-strip* **have** t=*Stuck*
  by (*auto dest*: *execn-Stuck-end*)
with *Stuck* **show** *?thesis*
  by *auto*

next
case (*Abrupt* s')
with *exec-strip* **have** t=*Abrupt* s'
  by (*auto dest*: *execn-Abrupt-end*)
with *Abrupt* **show** *?thesis*
  by *auto*

next
case (*Normal* s')
show *?thesis*
proof (*cases* $s' \in b$)
  case *True*
  with *Normal exec-strip*
  have $\Gamma \vdash \langle$*strip-guards F c1* ,*Normal* $s'\rangle$ =$n\Rightarrow$ t
    by (*auto elim*: *execn-Normal-elim-cases*)
  with *Normal True Cond.hyps* **obtain** t'
    where $\Gamma \vdash \langle$*c1*,*Normal* $s'\rangle$ =$n\Rightarrow$ t'
      isFault $t \longrightarrow$ *isFault* t'
      $t' \in$ *Fault* ‘ $(-F) \longrightarrow t'$=t
      $\neg$ *isFault* $t' \longrightarrow t' = t$
    by *blast*
  with *Normal True*
  show *?thesis*
    by (*blast intro*: *execn.intros*)
next
  case *False*
  with *Normal exec-strip*
  have $\Gamma \vdash \langle$*strip-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'
      $t' \in$ *Fault* ‘ $(-F) \longrightarrow t'$=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-strip*: $\Gamma\vdash\langle$*strip-guards F* (*While b 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'*)
    {
      fix *c' r w*
      assume *exec-c'*: $\Gamma\vdash\langle c',r\rangle$ =*n*$\Rightarrow$ *w*
      assume *c'*: *c'*=*While b* (*strip-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'* $\in$ *Fault* ‘ ($-F$) $\longrightarrow$ *w'*=*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* (*strip-guards F c*) **by** *fact*
        from *WhileTrue.hyps eqs*
        have *r-in-b*: *r*$\in$*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* ⟶ *isFault u′* **and**
    u′-Fault: *u′* ∈ *Fault* ‘ (−*F*) ⟶ *u′*=*u* **and**
    u′-noFault: ¬ *isFault u′* ⟶ *u′*=*u*
    by *blast*
  show *?thesis*
  proof (*cases isFault u′*)
    case *False*
    with *u′-noFault* **have** *u′*: *u′*=*u* **by** *simp*
    from *WhileTrue.hyps eqs* **obtain** *w′* **where**
      Γ⊢⟨*While b c*,*u*⟩ =*n*⇒ *w′*
      isFault w ⟶ *isFault w′*
      w′ ∈ *Fault* ‘ (−*F*) ⟶ *w′*=*w*
      ¬ *isFault w′* ⟶ *w′* = *w*
      by *blast*
    with *u′ exec-c r-in-b*
    show *?thesis*
      by (*blast intro*: *execn.WhileTrue*)
  next
    case *True*
    then obtain *f′* **where** *u′*: *u′*=*Fault f′*..
    with *exec-c r-in-b*
    have *exec*: Γ⊢⟨*While b c*,*Normal r*⟩ =*n*⇒ *Fault f′*
      by (*blast intro*: *execn.intros*)
    show *?thesis*
    proof (*cases isFault u*)
      case *True*
      then obtain *f* **where** *u*: *u*=*Fault f*..
      with *exec-strip-w* **have** *w*=*Fault f*
        by (*auto dest*: *execn-Fault-end*)
      with *exec u′ u u′-Fault*
      show *?thesis*
        by *auto*
    next
      case *False*
      with *u′-Fault u′* **have** *f′* ∈ *F*
        by (*cases u*) *auto*
      with *exec* **show** *?thesis*
        by *auto*
    qed
  qed
 qed
next
 case (*WhileFalse r b′ c″ n*)
 have *eqs*: *While b′ c″* = *While b* (*strip-guards F c*) **by** *fact*
 from *WhileFalse.hyps eqs*
 have *r-not-in-b*: *r*∉*b* **by** *simp*
 show *?case*
 proof −
  from *r-not-in-b*

```
      have Γ⊢⟨While b c,Normal r⟩ =n⇒ Normal r
        by (rule execn.WhileFalse)
      thus ?thesis
        by blast
    qed
  qed auto
} note hyp-while = this
show ?thesis
proof (cases s'∈b)
  case False
  with Normal exec-strip
  have t=s
    by (auto elim: execn-Normal-elim-cases)
  with Normal False show ?thesis
    by (auto intro: execn.intros)
next
  case True note s'-in-b = this
  with Normal exec-strip obtain r where
    exec-strip-c: 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 Γ⊢⟨While b c,r'⟩ =n⇒ Fault f'
      by auto
    with Normal s'-in-b exec-c
    have exec: Γ⊢⟨While b c,Normal s'⟩ =n⇒ Fault f'
      by (auto intro: execn.intros)
```

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

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

show *?case*
proof (*cases s*)
  case (*Fault f*)
  with *exec-strip* **have** *t=Fault f*
    by (*auto dest*: *execn-Fault-end*)
  with *Fault* **show** *?thesis*
    by *auto*
next
  case *Stuck*
  with *exec-strip* **have** *t=Stuck*
    by (*auto dest*: *execn-Stuck-end*)
  with *Stuck* **show** *?thesis*
    by *auto*
next
  case (*Abrupt s'*)
  with *exec-strip* **have** *t=Abrupt s'*
    by (*auto dest*: *execn-Abrupt-end*)
  with *Abrupt* **show** *?thesis*
    by *auto*
next
  case (*Normal s'*) **note** *s=this*
  with *exec-strip* **have**
   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 Γ⊢⟨c,s⟩ =n⇒ t
  shows ∃t'. Γ⊢⟨c,s⟩ =n⇒ t' ∧
                (isFault t ⟶ isFault t') ∧
                (t' ∈ Fault ` (− F) ⟶ t'=t) ∧
                (¬ isFault t' ⟶ 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: ¬ *isFault t'* ⟶ *t'* = *t*
    by *blast*
  show *?case*
  proof (*cases isFault r'*)
    case *True*
    then obtain *f'* **where** *r'*: *r'=Fault f'*..
    with *exec-c1* **have** Γ⊢⟨*Catch c1 c2*,*Normal s*⟩ =*n*⇒ *Fault f'*
      by (*auto intro*: *execn.intros*)
    with *r' r'-Fault* **show** *?thesis*
      by (*auto intro*: *execn.intros*)
  next
    case *False*
    with *r'-noFault* **have** *r'=Abrupt r* **by** *simp*
    with *exec-c1 exec-c2 t-Fault t'-noFault t'-Fault*
    show *?thesis*
      by (*blast intro*: *execn.intros*)
  qed
next
  case *CatchMiss* **thus** *?case* **by** (*fastforce intro*: *execn.intros elim*: *isFaultE*)
qed

lemma *exec-strip-guards-to-exec*:
  assumes *exec-strip*: Γ⊢⟨*strip-guards F c*,*s*⟩ ⇒ *t*
  shows ∃ *t'*. Γ⊢⟨*c*,*s*⟩ ⇒ *t'* ∧
            (*isFault t* ⟶ *isFault t'*) ∧
            (*t'* ∈ *Fault* ‘ (−*F*) ⟶ *t'=t*) ∧
            (¬ *isFault t'* ⟶ *t'=t*)
proof −
  from *exec-strip* **obtain** *n* **where**
    execn-strip: Γ⊢⟨*strip-guards F c*,*s*⟩ =*n*⇒ *t*
    by (*auto simp add*: *exec-iff-execn*)
  then obtain *t'* **where**
    Γ⊢⟨*c*,*s*⟩ =*n*⇒ *t'*
    isFault t ⟶ *isFault t' t'* ∈ *Fault* ‘ (−*F*) ⟶ *t'=t* ¬ *isFault t'* ⟶ *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* Γ⊢⟨*c*,*s*⟩ ⇒ *t*
  shows ∃ *t'*. Γ⊢⟨*c*,*s*⟩ ⇒ *t'* ∧
            (*isFault t* ⟶ *isFault t'*) ∧
            (*t'* ∈ *Fault* ‘ (−*F*) ⟶ *t'=t*) ∧
            (¬ *isFault t'* ⟶ *t'=t*)
proof −
  from *exec-strip* **obtain** *n* **where**
    execn-strip: *strip F* Γ⊢⟨*c*,*s*⟩ =*n*⇒ *t*
    by (*auto simp add*: *exec-iff-execn*)

then obtain t' **where**
$\Gamma\vdash\langle c,s\rangle =n\Rightarrow t'$
isFault $t \longrightarrow$ *isFault* t' $t' \in$ *Fault* ‘ $(-F) \longrightarrow t'=t$ $\neg$ *isFault* $t' \longrightarrow t'=t$
by (*blast dest*: *execn-strip-to-execn*)
thus *?thesis*
by (*blast intro*: *execn-to-exec*)
qed

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

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

lemma *execn-to-execn-strip*:
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
assumes *t-not-Fault*: $\neg$ *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** $(strip\ F\ \Gamma)\ p = Some\ (strip\text{-}guards\ F\ bdy)$
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*: $\Gamma\ p = Some\ bdy$ **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 $strip\ F\ \Gamma\vdash\langle strip\text{-}guards\ F\ bdy,Normal\ s\rangle =n\Rightarrow s'$
by *simp*
moreover from *bdy* **have** $(strip\ F\ \Gamma)\ p = Some\ (strip\text{-}guards\ F\ bdy)$
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 $\Gamma\vdash\langle \textit{strip-guards } F\ c,s\rangle =n\Rightarrow \textit{Fault } f$
  by *simp*
thus $\Gamma\vdash\langle \textit{strip-guards } F\ c,s\rangle \Rightarrow \textit{Fault } f$
  by (*rule execn-to-exec*)
qed

6.8 Lemmas about $c_1 \cap_g c_2$

lemma *inter-guards-execn-Normal-noFault*:
  $\bigwedge c\ c2\ s\ t\ n.\ [\![(c1 \cap_g c2) = \textit{Some } c;\ \Gamma\vdash\langle c,\textit{Normal } s\rangle =n\Rightarrow t;\ \neg\ \textit{isFault } t]\!]$
    $\Longrightarrow \Gamma\vdash\langle c1,\textit{Normal } s\rangle =n\Rightarrow t \wedge \Gamma\vdash\langle c2,\textit{Normal } s\rangle =n\Rightarrow t$
proof (*induct c1*)
  case *Skip*
  have $(\textit{Skip} \cap_g c2) = \textit{Some } c$ **by** *fact*
  then obtain *c2*: *c2=Skip* **and** *c*: *c=Skip*
    by (*simp add*: *inter-guards-Skip*)
  have $\Gamma\vdash\langle c,\textit{Normal } s\rangle =n\Rightarrow t$ **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 $(\textit{Basic } f \cap_g c2) = \textit{Some } c$ **by** *fact*
  then obtain *c2*: *c2=Basic f* **and** *c*: *c=Basic f*
    by (*simp add*: *inter-guards-Basic*)
  have $\Gamma\vdash\langle c,\textit{Normal } s\rangle =n\Rightarrow t$ **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 $(\textit{Spec } r \cap_g c2) = \textit{Some } c$ **by** *fact*
  then obtain *c2*: *c2=Spec r* **and** *c*: *c=Spec r*
    by (*simp add*: *inter-guards-Spec*)
  have $\Gamma\vdash\langle c,\textit{Normal } s\rangle =n\Rightarrow t$ **by** *fact*
  with *c* **have** $\Gamma\vdash\langle \textit{Spec } r,\textit{Normal } s\rangle =n\Rightarrow t$ **by** *simp*
  from *this Spec c2* **show** *?case*
    by (*cases*) (*auto intro*: *execn.intros*)
next
  case (*Seq a1 a2*)
  have *noFault*: $\neg$ *isFault t* **by** *fact*
  have $(\textit{Seq } a1\ a2 \cap_g c2) = \textit{Some } c$ **by** *fact*
  then obtain *b1 b2 d1 d2* **where**
    c2: *c2=Seq b1 b2* **and**
    d1: $(a1 \cap_g b1) = \textit{Some } d1$ **and** *d2*: $(a2 \cap_g b2) = \textit{Some } d2$ **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*: ¬ *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 $(While\ b\ bdy1 \cap_g c2) = Some\ c$ **by** *fact*
then obtain *bdy2 bdy* **where**
c2: *c2=While b bdy2* **and**
bdy: $(bdy1 \cap_g bdy2) = Some\ bdy$ **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
```

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

using *execn.DynCom* **by** *fastforce*
qed
next
case *Guard* **thus** *?case*
by (*fastforce elim*: *execn-Normal-elim-cases intro*: *execn.intros*
simp add: *inter-guards-Guard*)
next
case *Throw* **thus** *?case*
by (*fastforce elim*: *execn-Normal-elim-cases*
simp add: *inter-guards-Throw*)
next
case (*Catch a1 a2*)
have *noFault*: $\neg$ *isFault t* **by** *fact*
have (*Catch a1 a2* $\cap_g$ *c2*) = *Some c* **by** *fact*
then obtain *b1 b2 d1 d2* **where**
c2: *c2=Catch b1 b2* **and**
d1: (*a1* $\cap_g$ *b1*) = *Some d1* **and** *d2*: (*a2* $\cap_g$ *b2*) = *Some d2* **and**
c: *c=Catch d1 d2*
by (*auto simp add*: *inter-guards-Catch*)
have $\Gamma\vdash\langle c, Normal\ s\rangle =n\Rightarrow t$ **by** *fact*
with *c* **have** $\Gamma\vdash\langle Catch\ d1\ d2, Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
then show *?case*
proof (*cases*)
fix s'
assume $\Gamma\vdash\langle d1, Normal\ s\rangle =n\Rightarrow Abrupt\ s'$
with *d1 Catch.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
assume $\Gamma\vdash\langle d2, Normal\ s'\rangle =n\Rightarrow t$
with *d2 Catch.hyps noFault*
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 *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 ∩g c2) = Some c by fact
  then obtain b1 b2 d1 d2 where
    c2: c2=Seq b1 b2 and
    d1: (a1 ∩g b1) = Some d1 and d2: (a2 ∩g b2) = Some d2 and
    c: c=Seq d1 d2
    by (auto simp add: inter-guards-Seq)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c obtain s' where
    exec-d1: Γ⊢⟨d1,Normal s⟩ =n⇒ s' and
    exec-d2: Γ⊢⟨d2,s'⟩ =n⇒ Fault f
    by (auto elim: execn-Normal-elim-cases)
  show ?case
  proof (cases s')
    case (Fault f')
    with exec-d2 have f'=f
      by (auto dest: execn-Fault-end)
    with Fault d1 exec-d1
    have Γ⊢⟨a1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨b1,Normal s⟩ =n⇒ Fault f
      by (auto dest: Seq.hyps)
    thus ?thesis
    proof (cases rule: disjE [consumes 1])
      assume Γ⊢⟨a1,Normal s⟩ =n⇒ Fault f
      hence Γ⊢⟨Seq a1 a2,Normal s⟩ =n⇒ Fault f
        by (auto intro: execn.intros)
      thus ?thesis
        by simp
    next
      assume Γ⊢⟨b1,Normal s⟩ =n⇒ Fault f
      hence Γ⊢⟨Seq b1 b2,Normal s⟩ =n⇒ Fault f
        by (auto intro: execn.intros)
      with c2 show ?thesis
        by simp
    qed
  next
    case Abrupt with exec-d2 show ?thesis by (auto dest: execn-Abrupt-end)
  next
    case Stuck with exec-d2 show ?thesis by (auto dest: execn-Stuck-end)
  next
    case (Normal s'')
    with inter-guards-execn-noFault [OF d1 exec-d1] obtain
      exec-a1: Γ⊢⟨a1,Normal s⟩ =n⇒ Normal s'' and
```

```
      exec-b1: Γ⊢⟨b1,Normal s⟩ =n⇒ Normal s''
      by simp
    moreover from d2 exec-d2 Normal
    have Γ⊢⟨a2,Normal s''⟩ =n⇒ Fault f ∨ Γ⊢⟨b2,Normal s''⟩ =n⇒ Fault f
      by (auto dest: Seq.hyps)
    ultimately show ?thesis
      using c2 by (auto intro: execn.intros)
  qed
next
  case (Cond b t1 e1)
  have (Cond b t1 e1 ∩g c2) = Some c by fact
  then obtain t2 e2 t e where
    c2: c2=Cond b t2 e2 and
    t: (t1 ∩g t2) = Some t and
    e: (e1 ∩g e2) = Some e and
    c: c=Cond b t e
    by (auto simp add: inter-guards-Cond)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨Cond b t e,Normal s⟩ =n⇒ Fault f by simp
  thus ?case
  proof (cases)
    assume s ∈ b
    moreover assume Γ⊢⟨t,Normal s⟩ =n⇒ Fault f
    with t have Γ⊢⟨t1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨t2,Normal s⟩ =n⇒ Fault f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  next
    assume s ∉ b
    moreover assume Γ⊢⟨e,Normal s⟩ =n⇒ Fault f
    with e have Γ⊢⟨e1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨e2,Normal s⟩ =n⇒ Fault f
      by (auto dest: Cond.hyps)
    ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
  qed
next
  case (While b bdy1)
  have (While b bdy1 ∩g c2) = Some c by fact
  then obtain bdy2 bdy where
    c2: c2=While b bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=While b bdy
    by (auto simp add: inter-guards-While)
  have exec-c: Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  {
    fix s t n w w1 w2
    assume exec-w: Γ⊢⟨w,Normal s⟩ =n⇒ t
    assume w: w=While b bdy
    assume Fault: t=Fault f
    from exec-w w Fault
    have Γ⊢⟨While b bdy1,Normal s⟩ =n⇒ Fault f∨
```

$\Gamma\vdash\langle While\ b\ bdy2,Normal\ s\rangle =n\Rightarrow Fault\ f$
proof (*induct*)
case (*WhileTrue s b′ bdy′ n s′ s′′*)
have *eqs*: *While b′ bdy′ = While b bdy* **by** *fact*
from *WhileTrue* **have** *s-in-b*: $s \in b$ **by** *simp*
have *Fault-s′′*: $s''=Fault\ f$ **by** *fact*
from *WhileTrue*
have *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ s\rangle =n\Rightarrow s'$ **by** *simp*
from *WhileTrue*
have *exec-w*: $\Gamma\vdash\langle While\ b\ bdy,s'\rangle =n\Rightarrow s''$ **by** *simp*
show *?case*
proof (*cases s′*)
case (*Fault f′*)
with *exec-w Fault-s′′* **have** $f'=f$
by (*auto dest*: *execn-Fault-end*)
with *Fault exec-bdy bdy While.hyps*
have $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Fault\ f \lor \Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Fault\ f$
by *auto*
with *s-in-b* **show** *?thesis*
by (*fastforce intro*: *execn.intros*)
next
case (*Normal s′′′*)
with *inter-guards-execn-noFault* [*OF bdy exec-bdy*]
obtain $\Gamma\vdash\langle bdy1,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
$\Gamma\vdash\langle bdy2,Normal\ s\rangle =n\Rightarrow Normal\ s'''$
by *auto*
moreover
from *Normal WhileTrue*
have $\Gamma\vdash\langle While\ b\ bdy1,Normal\ s'''\rangle =n\Rightarrow Fault\ f \lor$
$\Gamma\vdash\langle While\ b\ bdy2,Normal\ s'''\rangle =n\Rightarrow Fault\ f$
by *simp*
ultimately show *?thesis*
using *s-in-b* **by** (*fastforce intro*: *execn.intros*)
next
case (*Abrupt s′′′*)
with *exec-w Fault-s′′* **show** *?thesis* **by** (*fastforce dest*: *execn-Abrupt-end*)
next
case *Stuck*
with *exec-w Fault-s′′* **show** *?thesis* **by** (*fastforce dest*: *execn-Stuck-end*)
qed
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.intros*)
qed (*simp-all*)
}
with *this* [*OF exec-c c*] *c2*
show *?case*
by *auto*
next
case *Call* **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Call*)

```
next
  case (DynCom f1)
  have (DynCom f1 ∩g c2) = Some c by fact
  then obtain f2 where
    c2: c2=DynCom f2 and
    F-defined: ∀ s. ((f1 s) ∩g (f2 s)) ≠ None and
    c: c=DynCom (λs. the ((f1 s) ∩g (f2 s)))
    by (auto simp add: inter-guards-DynCom)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨DynCom (λs. the ((f1 s) ∩g (f2 s))),Normal s⟩ =n⇒ Fault f
by simp
  then show ?case
  proof (cases)
    assume exec-F: Γ⊢⟨the (f1 s ∩g f2 s),Normal s⟩ =n⇒ Fault f
    from F-defined obtain F where (f1 s ∩g f2 s) = Some F
      by auto
    with DynCom.hyps this exec-F c2
    show ?thesis
      by (fastforce intro: execn.intros)
  qed
next
  case (Guard m g1 bdy1)
  have (Guard m g1 bdy1 ∩g c2) = Some c by fact
  then obtain g2 bdy2 bdy where
    c2: c2=Guard m g2 bdy2 and
    bdy: (bdy1 ∩g bdy2) = Some bdy and
    c: c=Guard m (g1 ∩ g2) bdy
    by (auto simp add: inter-guards-Guard)
  have Γ⊢⟨c,Normal s⟩ =n⇒ Fault f by fact
  with c have Γ⊢⟨Guard m (g1 ∩ g2) bdy,Normal s⟩ =n⇒ Fault f
    by simp
  thus ?case
  proof (cases)
    assume f-m: Fault f = Fault m
    assume s ∉ g1 ∩ g2
    hence s∉g1 ∨ s∉g2
      by blast
    with c2 f-m show ?thesis
      by (auto intro: execn.intros)
  next
    assume s ∈ g1 ∩ g2
    moreover
    assume Γ⊢⟨bdy,Normal s⟩ =n⇒ Fault f
    with bdy have Γ⊢⟨bdy1,Normal s⟩ =n⇒ Fault f ∨ Γ⊢⟨bdy2,Normal s⟩ =n⇒
Fault f
      by (rule Guard.hyps)
    ultimately show ?thesis
      using c2
      by (auto intro: execn.intros)
```

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


lemma inter-guards-execn-Fault:
  assumes c: (c1 ∩g c2) = Some c
  assumes exec-c: Γ⊢⟨c,s⟩ =n⇒ Fault f
  shows Γ⊢⟨c1,s⟩ =n⇒ Fault f ∨ Γ⊢⟨c2,s⟩ =n⇒ Fault f
proof (cases s)
  case (Fault f)
  with exec-c show ?thesis
    by (auto dest: execn-Fault-end)
next
  case (Abrupt s')
  with exec-c show ?thesis
```

by (*fastforce dest*: *execn-Abrupt-end*)
next
case *Stuck*
with *exec-c* **show** *?thesis*
by (*fastforce dest*: *execn-Stuck-end*)
next
case (*Normal s'*)
with *exec-c inter-guards-execn-Normal-Fault* [*OF c*]
show *?thesis*
by *blast*
qed

lemma *inter-guards-exec-Fault*:
assumes *c*: $(c1 \cap_g c2) = Some\ c$
assumes *exec-c*: $\Gamma\vdash\langle c,s\rangle \Rightarrow Fault\ f$
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

6.9 Restriction of Procedure Environment

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

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

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

lemma *exec-restrict-to-exec*:
assumes *exec-restrict*: $\Gamma|_A\vdash\langle c,s\rangle \Rightarrow t$
assumes *notStuck*: $t\neq Stuck$
shows $\Gamma\vdash\langle c,s\rangle \Rightarrow t$

using *exec-restrict notStuck*
by (*induct*) (*auto intro*: *exec.intros dest*: *restrict-SomeD Stuck-end*)

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

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

lemma *execn-to-execn-restrict*:
  assumes *execn*: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$
  shows $\exists t'.\ \Gamma|_P\vdash\langle c,s\rangle =n\Rightarrow t' \wedge (t=Stuck \longrightarrow t'=Stuck) \wedge$
        $(\forall f.\ t=Fault\ f \longrightarrow t'\in\{Fault\ f,Stuck\}) \wedge (t'\neq Stuck \longrightarrow t'=t)$
using *execn*
proof (*induct*)
  case *Skip* **show** *?case* **by** (*blast intro*: *execn.Skip*)
next
  case *Guard* **thus** *?case* **by** (*auto intro*: *execn.Guard*)
next
  case *GuardFault* **thus** *?case* **by** (*auto intro*: *execn.GuardFault*)
next
  case *FaultProp* **thus** *?case* **by** (*auto intro*: *execn.FaultProp*)
next
  case *Basic* **thus** *?case* **by** (*auto intro*: *execn.Basic*)
next
  case *Spec* **thus** *?case* **by** (*auto intro*: *execn.Spec*)
next
  case *SpecStuck* **thus** *?case* **by** (*auto intro*: *execn.SpecStuck*)
next
  case *Seq* **thus** *?case* **by** (*metis insertCI execn.Seq StuckProp*)
next
  case *CondTrue* **thus** *?case* **by** (*auto intro*: *execn.CondTrue*)
next
  case *CondFalse* **thus** *?case* **by** (*auto intro*: *execn.CondFalse*)
next
  case *WhileTrue* **thus** *?case* **by** (*metis insertCI execn.WhileTrue StuckProp*)
next
  case *WhileFalse* **thus** *?case* **by** (*auto intro*: *execn.WhileFalse*)
next
  case (*Call p bdy n s s'*)
  have $\Gamma\ p = Some\ bdy$ **by** *fact*
  show *?case*
  proof (*cases* $p \in P$)
    case *True*
    with *Call* **have** $(\Gamma|_P)\ p = Some\ bdy$

by (*simp*)
with *Call* **show** *?thesis*
by (*auto intro*: *execn.intros*)
next
case *False*
hence $(\Gamma|_P)$ p = *None* **by** *simp*
thus *?thesis*
by (*auto intro*: *execn.CallUndefined*)
qed
next
case (*CallUndefined p n s*)
have 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*: $\neg$ *isAbr w* **by** *fact*
  show *?case*
  proof (*cases w'*)
    case (*Normal s'*)
    with *w'-noStuck* **have** $w'=w$
      by *simp*
    with *exec-c1' Normal w-Stuck w-Fault w'-noStuck*
    show *?thesis*
      by (*fastforce intro*: *execn.CatchMiss*)
  next
    case (*Abrupt s'*)
    with *w'-noStuck* **have** $w'=w$
      by *simp*
    with *noAbr-w Abrupt* **show** *?thesis* **by** *simp*
  next
    case (*Fault f*)
    with *w'-noStuck* **have** $w'=w$
      by *simp*
    with *exec-c1' Fault w-Stuck w-Fault w'-noStuck*
    show *?thesis*
      by (*fastforce intro*: *execn.CatchMiss*)
  next
    case *Stuck*
    with *exec-c1' w-Stuck w-Fault w'-noStuck*
    show *?thesis*
      by (*fastforce intro*: *execn.CatchMiss*)
  qed
qed

lemma *exec-to-exec-restrict*:
  assumes *exec*: $\Gamma\vdash\langle c,s\rangle \Rightarrow t$
  shows $\exists t'.\ \Gamma|_P\vdash\langle c,s\rangle \Rightarrow t' \wedge (t=Stuck \longrightarrow t'=Stuck) \wedge$
    $(\forall f.\ t=Fault\ f \longrightarrow t'\in\{Fault\ f,Stuck\}) \wedge (t'\neq Stuck \longrightarrow t'=t)$
proof −
  from *exec* **obtain** *n* **where**
    execn-strip: $\Gamma\vdash\langle c,s\rangle =n\Rightarrow t$

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

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

lemma *notStuck-SeqD1*:
$[\![\Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin\{Stuck\}]\!] \Longrightarrow \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*auto simp add*: *final-notin-def dest*: *exec.Seq*)

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

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

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

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

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

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

lemma *notStuck-CallD*:

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

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

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

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

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

6.10 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*: ¬ *isFault s*
shows ¬ *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*: ¬ *isFault s*
shows ¬ *isFault t*
using *exec noguards-c s-no-Fault*
proof (*induct*)

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

end

7 Hoare Logic for Partial Correctness

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

type-synonym (*'s*,*'p*) *quadruple* = (*'s assn* × *'p* × *'s assn* × *'s assn*)

7.1 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*] ⇒ *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)

7.2 Properties of Validity

lemma *valid-iff-nvalid*: $\Gamma \models_{/F} P\ c\ Q,A = (\forall n.\ \Gamma \models n{:}_{/F} P\ c\ Q,A)$
apply (*simp only*: *valid-def nvalid-def exec-iff-execn*)
apply (*blast dest*: *exec-final-notin-to-execn*)
done

lemma *cnvalid-to-cvalid*: $(\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c\ Q,A) \Longrightarrow \Gamma,\Theta \models_{/F}\ P\ c\ Q,A$
apply (*unfold cvalid-def cnvalid-def valid-iff-nvalid* [*THEN eq-reflection*])
apply *fast*
done

lemma *nvalidI*:
$[\![\bigwedge s\ t.\ [\![\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow t;s \in P;\ t\notin\ Fault\ `\ F]\!] \Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A]\!]$
$\Longrightarrow \Gamma\models n{:}_{/F}\ P\ c\ Q,A$
by (*auto simp add*: *nvalid-def*)

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

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

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

lemma *cnvalidI*:
$[\![\bigwedge s\ t.\ [\![\forall (P,p,Q,A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A;$
$\Gamma\vdash\langle c,Normal\ s\ \rangle =n\Rightarrow t;s \in P;t\notin Fault\ `\ F]\!]$
$\Longrightarrow t \in Normal\ `\ Q \cup Abrupt\ `\ A]\!]$
$\Longrightarrow \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
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': $t' \in Fault\ `\ (-F') \longrightarrow t'=t\ \neg\ isFault\ t' \longrightarrow t'=t$
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* ‘ $(-F') \longrightarrow t'=t$ $\neg$ *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

7.3 The Hoare Rules: $\Gamma,\Theta \vdash_{/F} P\ c\ Q,A$

lemma *mono-WeakenContext*: $A \subseteq B \Longrightarrow$
$(\lambda(P,\ c,\ Q,\ A').\ (\Gamma,\ \Theta,\ F,\ P,\ c,\ Q,\ A') \in A)\ x \longrightarrow$
$(\lambda(P,\ c,\ Q,\ A').\ (\Gamma,\ \Theta,\ F,\ P,\ c,\ Q,\ A') \in B)\ x$

apply *blast*
done

inductive *hoarep*::[('s,'p,'f) body,('s,'p) quadruple set,'f set,
'*s assn*,('s,'p,'f) com, 's assn,'s assn] => bool
((3-,-/⊢'/- (-/ (-)/ -,/-)) [60,60,60,1000,20,1000,1000]60)
for Γ::('s,'p,'f) *body*
where
Skip: $\Gamma,\Theta\vdash_{/F}$ *Q Skip Q*,*A*

| *Basic*: $\Gamma,\Theta\vdash_{/F}$ $\{s.\ f\ s \in Q\}$ (*Basic f*) *Q*,*A*

| *Spec*: $\Gamma,\Theta\vdash_{/F}$ $\{s.\ (\forall\, t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists\, t.\ (s,t) \in r)\}$ (*Spec r*) *Q*,*A*

| *Seq*: $[\![\Gamma,\Theta\vdash_{/F} P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (Seq\ c_1\ c_2)\ Q,A$

| *Cond*: $[\![\Gamma,\Theta\vdash_{/F} (P \cap b)\ c_1\ Q,A;\ \Gamma,\Theta\vdash_{/F} (P \cap -\ b)\ c_2\ Q,A]\!]$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (Cond\ b\ c_1\ c_2)\ Q,A$

| *While*: $\Gamma,\Theta\vdash_{/F} (P \cap b)\ c\ P,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (While\ b\ c)\ (P \cap -\ b),A$

| *Guard*: $\Gamma,\Theta\vdash_{/F} (g \cap P)\ c\ Q,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} (g \cap P)\ (Guard\ f\ g\ c)\ Q,A$

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

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

| *DynCom*:
$\forall\, s \in P.\ \Gamma,\Theta\vdash_{/F} P\ (c\ s)\ Q,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{/F} P\ (DynCom\ c)\ Q,A$

| *Throw*: $\Gamma,\Theta\vdash_{/F} A\ Throw\ Q,A$

| *Catch*: $[\![\Gamma,\Theta\vdash_{/F} P\ c_1\ Q,R;\ \Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A]\!] \Longrightarrow\ \Gamma,\Theta\vdash_{/F} P\ Catch\ c_1\ c_2\ Q,A$

| *Conseq*: $\forall s \in P.\ \exists P'\ Q'\ A'.\ \Gamma,\Theta \vdash_{/F} P'\ c\ Q',A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$
$\Longrightarrow \Gamma,\Theta \vdash_{/F} P\ c\ Q,A$

| *Asm*: $[\![(P,p,Q,A) \in \Theta]\!]$
$\Longrightarrow$
$\Gamma,\Theta \vdash_{/F} P\ (\mathit{Call}\ p)\ Q,A$

| *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*)+

7.4 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*: $[\![\forall Z.\ \Gamma,\Theta \vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\quad \forall s.\ s \in P \longrightarrow (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\quad \Longrightarrow$
$\quad \Gamma,\Theta{\vdash}_{/F}\ P\ c\ Q,A$
by (*rule Conseq*) *blast*

theorem *conseqPrePost* [*trans*]:
$\Gamma,\Theta{\vdash}_{/F}\ P'\ c\ Q',A' \Longrightarrow P \subseteq P' \Longrightarrow\ Q' \subseteq Q \Longrightarrow A' \subseteq A \Longrightarrow\ \Gamma,\Theta{\vdash}_{/F}\ P\ c\ Q,A$
by (*rule conseq* [**where** $?P'$=$\lambda Z.\ P'$ **and** $?Q'$=$\lambda Z.\ Q'$]) *auto*

lemma *conseqPre* [*trans*]: $\Gamma,\Theta{\vdash}_{/F}\ P'\ c\ Q,A \Longrightarrow P \subseteq P' \Longrightarrow \Gamma,\Theta{\vdash}_{/F}\ P\ c\ Q,A$
by (*rule conseq*) *auto*

lemma *conseqPost* [*trans*]: $\Gamma,\Theta{\vdash}_{/F}\ P\ c\ Q',A' \Longrightarrow Q' \subseteq Q \Longrightarrow A' \subseteq A$
$\Longrightarrow\ \ \Gamma,\Theta{\vdash}_{/F}\ P\ c\ Q,A$
by (*rule conseq*) *auto*

lemma *CallRec'*:
$[\![p{\in}Procs;\ Procs \subseteq dom\ \Gamma;$
$\forall p{\in}Procs.$
$\quad \forall Z.\ \Gamma,\Theta \cup (\bigcup p{\in}Procs.\ \bigcup Z.\ \{((P\ p\ Z),p,Q\ p\ Z,A\ p\ Z)\})$
$\qquad \vdash_{/F} (P\ p\ Z)\ (the\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)]\!]$
$\Longrightarrow$
$\Gamma,\Theta{\vdash}_{/F}\ (P\ p\ Z)\ (Call\ p)\ (Q\ p\ Z),(A\ p\ Z)$
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

8 Properties of Partial Correctness Hoare Logic

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

8.1 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 $\Gamma,\Theta \models n{:}_{/F}\ P\ \mathit{Skip}\ P,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume $\Gamma\vdash\langle \mathit{Skip},\mathit{Normal}\ s\rangle =n\Rightarrow\ t\ s \in P$
    thus $t \in \mathit{Normal}$ ‘ $P \cup \mathit{Abrupt}$ ‘ A
      by *cases auto*
  qed
next
  case (*Basic* Θ *F f P A*)
  show $\Gamma,\Theta \models n{:}_{/F}\ \{s.\ f\ s \in P\}\ (\mathit{Basic}\ f)\ P,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume $\Gamma\vdash\langle \mathit{Basic}\ f,\mathit{Normal}\ s\rangle =n\Rightarrow\ t\ s \in \{s.\ f\ s \in P\}$
    thus $t \in \mathit{Normal}$ ‘ $P \cup \mathit{Abrupt}$ ‘ A
      by *cases auto*
  qed
next
  case (*Spec* Θ *F r Q A*)
  show $\Gamma,\Theta\models n{:}_{/F}\ \{s.\ (\forall t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,\ t) \in r)\}\ \mathit{Spec}\ r\ Q,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume *exec*: $\Gamma\vdash\langle \mathit{Spec}\ r,\mathit{Normal}\ s\rangle =n\Rightarrow\ t$
    assume *P*: $s \in \{s.\ (\forall t.\ (s,\ t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,\ t) \in r)\}$
    from *exec P*
    show $t \in \mathit{Normal}$ ‘ $Q \cup \mathit{Abrupt}$ ‘ 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 $\Gamma,\Theta \models n{:}_{/F}\ P\ \mathit{Seq}\ c1\ c2\ Q,A$
  proof (*rule cnvalidI*)
    fix *s t*
    assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
    assume *exec*: $\Gamma\vdash\langle \mathit{Seq}\ c1\ c2,\mathit{Normal}\ s\rangle =n\Rightarrow\ t$
    assume *t-notin-F*: $t \notin \mathit{Fault}$ ‘ F
    assume *P*: $s \in P$
    from *exec P* **obtain** *r* **where**
      exec-c1: $\Gamma\vdash\langle c1,\mathit{Normal}\ s\rangle =n\Rightarrow\ r$ **and** *exec-c2*: $\Gamma\vdash\langle c2,r\rangle =n\Rightarrow\ t$
      by *cases auto*
    with *t-notin-F* **have** $r \notin \mathit{Fault}$ ‘ F
      by (*auto dest*: *execn-Fault-end*)
    with *valid-c1 ctxt exec-c1 P*

have *r*: *r*∈*Normal* ' *R* ∪ *Abrupt* ' *A*
  by (*rule cnvalidD*)
show *t*∈*Normal* ' *Q* ∪ *Abrupt* ' *A*
proof (*cases r*)
  case (*Normal r'*)
  with *exec-c2 r*
  show *t*∈*Normal* ' *Q* ∪ *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*: Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ =*n*⇒ *t*
  assume *P*: *s* ∈ *P*
  assume *t-notin-F*: *t* ∉ *Fault* ' *F*
  show *t* ∈ *Normal* ' *Q* ∪ *Abrupt* ' *A*
  proof (*cases s*∈*b*)
    case *True*
    with *exec* **have** Γ⊢⟨*c1*,*Normal s*⟩ =*n*⇒ *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** Γ⊢⟨*c2*,*Normal s*⟩ =*n*⇒ *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 $\Gamma,\Theta \models n{:}_{/F}\ P\ \mathit{While}\ b\ c\ (P \cap -\ b),A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
assume *exec*: $\Gamma \vdash \langle \mathit{While}\ b\ c,\mathit{Normal}\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin \mathit{Fault}\ `\ F$
show $t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}\ `\ A$
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*: $d{=}\mathit{While}\ b\ c$
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
from *exec d ctxt*
have $[\![s \in \mathit{Normal}\ `\ P;\ t \notin \mathit{Fault}\ `\ F]\!]$
$\Longrightarrow t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}`A$
proof (*induct*)
case (*WhileTrue* $s\ b'\ c'\ n\ r\ t$)
have *t-notin-F*: $t \notin \mathit{Fault}\ `\ F$ **by** *fact*
have *eqs*: $\mathit{While}\ b'\ c' = \mathit{While}\ b\ c$ **by** *fact*
note *valid-c*
moreover have *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$ **by** *fact*
moreover from *WhileTrue*
obtain $\Gamma \vdash \langle c,\mathit{Normal}\ s\rangle =n\Rightarrow r$ **and**
$\Gamma \vdash \langle \mathit{While}\ b\ c,r\rangle =n\Rightarrow t$ **and**
$\mathit{Normal}\ s \in \mathit{Normal}\ `(P \cap b)$ **by** *auto*
moreover with *t-notin-F* **have** $r \notin \mathit{Fault}\ `\ F$
by (*auto dest*: *execn-Fault-end*)
ultimately
have *r*: $r \in \mathit{Normal}\ `\ P \cup \mathit{Abrupt}\ `\ A$
by $-$ (*rule cnvalidD*,*auto*)
from *this - ctxt*
show $t \in \mathit{Normal}\ `\ (P \cap -\ b) \cup \mathit{Abrupt}\ `\ A$
proof (*cases r*)
case (*Normal* r')
with *r ctxt eqs t-notin-F*
show *?thesis*
by $-$ (*rule WhileTrue.hyps*,*auto*)
next
case (*Abrupt* r')
have $\Gamma \vdash \langle \mathit{While}\ b'\ c',r\rangle =n\Rightarrow t$ **by** *fact*

```
          with Abrupt have t=r
            by (auto dest: execn-Abrupt-end)
          with r Abrupt show ?thesis
            by blast
        next
          case Fault with r show ?thesis by blast
        next
          case Stuck with r show ?thesis by blast
        qed
      qed auto
    }
    with exec ctxt P t-notin-F
    show ?thesis
      by auto
  next
    case False
    with exec P have t=Normal s
      by cases auto
    with P False
    show ?thesis
      by auto
  qed
qed
next
  case (Guard Θ F g P c Q A f)
  have valid-c: ⋀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: ⋀n. Γ,Θ ⊨n:/F (g ∩ P) c Q,A by fact
  have f-F: f ∈ F by fact
  show Γ,Θ ⊨n:/F 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*: $t \notin$ *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 ∀(P,p,Q,A) ∈Specs. Γ ⊨Suc m:/F P (Call p) Q,A
        proof (clarify)
          fix P p Q A assume p: (P,p,Q,A) ∈ Specs
          show Γ ⊨Suc m:/F P (Call p) Q,A
          proof (rule nvalidI)
            fix s t
            assume exec-call:
              Γ⊢⟨Call p,Normal s⟩ =Suc m⇒ t
            assume Pre: s ∈ P
            assume t-notin-F: t ∉ Fault ‘ F
            from exec-call
            show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
            proof (cases)
              fix bdy m′
              assume m: Suc m = Suc m′
              assume bdy: Γ p = Some bdy
              assume exec-body: Γ⊢⟨bdy,Normal s⟩ =m′⇒ t
              from Pre valid-body-m exec-body bdy m p t-notin-F
              show ?thesis
                by (fastforce simp add: nvalid-def)
            next
              assume Γ p = None
              with valid-body p have False by auto
              thus ?thesis ..
            qed
          qed
        qed
      qed
    }
    with p show ?thesis
      by (fastforce simp add: cnvalid-def)
  qed
next
  case (DynCom P Θ F c Q A)
  hence valid-c: ∀s∈P. (∀n. Γ,Θ⊨n:/F P (c s) Q,A) by auto
  show Γ,Θ⊨n:/F P DynCom 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⊢⟨DynCom c,Normal s⟩ =n⇒ t
    assume P: s ∈ P
    assume t-notin-Fault: t ∉ Fault ‘ F
    from exec show t ∈ Normal ‘ Q ∪ Abrupt ‘ A
    proof (cases)
      assume Γ⊢⟨c s,Normal s⟩ =n⇒ 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 $\Gamma,\Theta \models n{:}_{/F}\ A\ \mathit{Throw}\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume $\Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ s\rangle =n\Rightarrow t\ s \in A$
then show $t \in \mathit{Normal}\ \text{`}\ Q \cup \mathit{Abrupt}\ \text{`}\ A$
by *cases simp*
qed
next
case (*Catch* Θ *F P* c_1 *Q R* c_2 *A*)
have *valid-c1*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ c_1\ Q,R$ **by** *fact*
have *valid-c2*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ R\ c_2\ Q,A$ **by** *fact*
show $\Gamma,\Theta \models n{:}_{/F}\ P\ \mathit{Catch}\ c_1\ c_2\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-Fault*: $t \notin \mathit{Fault}\ \text{`}\ F$
from *exec* **show** $t \in \mathit{Normal}\ \text{`}\ Q \cup \mathit{Abrupt}\ \text{`}\ A$
proof (*cases*)
fix s'
assume *exec-c1*: $\Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle =n\Rightarrow \mathit{Abrupt}\ s'$
assume *exec-c2*: $\Gamma\vdash\langle c_2,\mathit{Normal}\ s'\rangle =n\Rightarrow t$
from *cnvalidD* [*OF valid-c1 ctxt exec-c1 P*]
have $\mathit{Abrupt}\ s' \in \mathit{Abrupt}\ \text{`}\ R$
by *auto*
with *cnvalidD* [*OF valid-c2 ctxt - - t-notin-Fault*] *exec-c2*
show *?thesis*
by *fastforce*
next
assume *exec-c1*: $\Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle =n\Rightarrow t$
assume *notAbr*: $\neg$ *isAbr t*
from *cnvalidD* [*OF valid-c1 ctxt exec-c1 P t-notin-Fault*]
have $t \in \mathit{Normal}\ \text{`}\ Q \cup \mathit{Abrupt}\ \text{`}\ R$ **.**
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*:$\forall (P, p, Q, A){\in}\Theta.\ \Gamma{\models}n{:}_{/F}\ P\ (Call\ p)\ Q{,}A$
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 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 $t \in Normal\ `\ Q' \cup Abrupt\ `\ A'$
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 $\Gamma{,}\Theta\ {\models}n{:}_{/F}\ P\ (Call\ p)\ Q{,}A$
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*: $\Gamma{\vdash}\langle Call\ p{,}Normal\ s\rangle\ {=}n{\Rightarrow}\ t$
from *asm ctxt* **have** $\Gamma\ {\models}n{:}_{/F}\ P\ Call\ p\ Q{,}A$ **by** *auto*
moreover
assume $s \in P\ t \notin Fault\ `\ F$
ultimately
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
using *exec*
by (*auto simp add*: *nvalid-def*)
qed
next
case *ExFalso* **thus** *?case* **by** *iprover*
qed

theorem *hoare-sound*: $\Gamma{,}\Theta{\vdash}_{/F}\ P\ c\ Q{,}A \Longrightarrow \Gamma{,}\Theta{\models}_{/F}\ P\ c\ Q{,}A$
by (*iprover intro*: *cnvalid-to-cvalid hoare-cnvalid*)

8.2 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 $\Gamma,\{\} \vdash_{/F} \{s.\ s=Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
using *MGT* [*rule-format*, *of Z*]
apply (*rule conseqPrePost*)
apply (*insert valid*)
apply (*fastforce simp add*: *valid-def final-notin-def*)
apply (*fastforce simp add*: *valid-def*)
apply (*fastforce simp add*: *valid-def*)
done

Semantic equivalence of both kind of formulations

lemma *valid-involved-to-valid*:
  assumes *valid*:
    $\forall Z.\ \Gamma \models_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
  shows $\Gamma \models_{/F} P\ c\ Q,A$
  using *valid*
  apply (*simp add*: *valid-def*)
  apply *clarsimp*
  apply (*erule-tac x=x* **in** *allE*)
  apply (*erule-tac x=Normal x* **in** *allE*)
  apply (*erule-tac x=t* **in** *allE*)
  apply *fastforce*
  done

The sophisticated consequence rule allow us to do this semantical transformation on the hoare-level, too. The magic is, that it allow us to choose the instance of Z under the assumption of an state $s \in P$

lemma
  assumes *deriv*:
    $\forall Z.\ \Gamma,\{\} \vdash_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
  shows $\Gamma,\{\} \vdash_{/F} P\ c\ Q,A$
  apply (*rule ConseqMGT* [*OF deriv*])
  apply *auto*
  done

lemma *valid-to-valid-involved*:
  $\Gamma \models_{/F} P\ c\ Q,A \Longrightarrow$
    $\Gamma \models_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
by (*simp add*: *valid-def Collect-conv-if*)

lemma
  assumes *deriv*: $\Gamma,\{\} \vdash_{/F} P\ c\ Q,A$
  shows $\Gamma,\{\} \vdash_{/F} \{s.\ s{=}Z \wedge s \in P\}\ c\ \{t.\ Z \in P \longrightarrow t \in Q\},\{t.\ Z \in P \longrightarrow t \in A\}$
  apply (*rule conseqPrePost* [*OF deriv*])
  apply *auto*
  done

lemma *conseq-extract-state-indep-prop*:
  assumes *state-indep-prop*: $\forall s \in P.\ R$
  assumes *to-show*: $R \Longrightarrow \Gamma,\Theta \vdash_{/F} P\ c\ Q,A$
  shows $\Gamma,\Theta \vdash_{/F} P\ c\ Q,A$
  apply (*rule Conseq*)
  apply (*clarify*)
  apply (*rule-tac x=P* **in** *exI*)
  apply (*rule-tac x=Q* **in** *exI*)

apply (*rule-tac* $x=A$ **in** *exI*)
using *state-indep-prop to-show*
by *blast*

lemma *MGT-lemma*:
assumes *MGT-Calls*:
$\forall p{\in}dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ ({-}F))\}$
$(Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
shows $\bigwedge Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ ({-}F))\}$
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 $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Skip,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ ({-}F))\}$
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 $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Basic\ f,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ ({-}F))\}$
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 $\Gamma,\Theta\vdash_{/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Spec\ r,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ ({-}F))\}$
Spec r
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Spec\ r,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoarep.Spec* [*THEN conseqPre*])
apply (*clarsimp simp add*: *final-notin-def*)
apply (*case-tac* $\exists t.\ (Z,t) \in r$)
apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
done
next
case (*Seq c1 c2*)
have *hyp-c1*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}$
$({-}F))\}\ c1$
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Seq.hyps* **by** *iprover*
have *hyp-c2*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}$

$(-F))\}$ *c2*
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Seq.hyps* **by** *iprover*
from *hyp-c1*
have $\Gamma,\Theta\vdash_{/F}\{s.\ s=Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\ \wedge$
$\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT*)
(*auto dest*: *Seq-NoFaultStuckD1* [*simplified*] *Seq-NoFaultStuckD2* [*simplified*]
intro: *exec.Seq*)
thus $\Gamma,\Theta\vdash_{/F}\{s.\ s=Z \wedge \Gamma\vdash\langle Seq\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$

Seq c1 c2
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule hoarep.Seq*)
show $\Gamma,\Theta\vdash_{/F}\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\ \wedge$
$\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
c2
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
fix *r t*
assume $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ r$ $\Gamma\vdash\langle c2,Normal\ r\rangle \Rightarrow Normal\ t$
then show $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*iprover intro*: *exec.intros*)
next
fix *r t*
assume $\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ r$ $\Gamma\vdash\langle c2,Normal\ r\rangle \Rightarrow Abrupt\ t$
then show $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*iprover intro*: *exec.intros*)
qed
qed
next
case (*Cond b c1 c2*)
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}\{s.\ s=Z \wedge \Gamma\vdash\langle c1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
c1
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *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: $final\text{-}notin\text{-}def$)
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: $final\text{-}notin\text{-}def$)
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 $\forall Z.\ \Gamma,\Theta\vdash_{/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$ **by** $iprover$
then show $\Gamma,\Theta\vdash_{/F} (?P'\ Z \cap b)\ c\ (?P'\ Z),(?A'\ Z)$
proof ($rule$ $ConseqMGT$)
fix s

assume $s \in \{t.\ (Z,\ t) \in \ ?unroll\ \wedge$
$(\forall\, e.\ (Z,e) \in ?unroll \longrightarrow e \in b$
$\longrightarrow \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))\ \wedge$
$(\forall\, u.\ \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow\ Abrupt\ u))\}$
$\cap\ b$
then obtain
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**
s-in-b: $s \in b$
by *blast*
show $s \in \{t.\ t = s\ \wedge\ \Gamma \vdash \langle c, Normal\ t\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))\}\ \wedge$
$(\forall\, t.\ t \in \{t.\ \Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow\ Normal\ t\} \longrightarrow$
$t \in \{t.\ (Z,\ t) \in \ ?unroll\ \wedge$
$(\forall\, e.\ (Z,e) \in ?unroll \longrightarrow\ \ e \in b$
$\longrightarrow \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))\ \wedge$
$(\forall\, u.\ \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow\ Abrupt\ u))\})\ \wedge$
$(\forall\, t.\ t \in \{t.\ \Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow\ Abrupt\ t\} \longrightarrow$
$t \in \{t.\ \Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow\ Abrupt\ t\})$
(**is** *?C1* $\wedge$ *?C2* $\wedge$ *?C3*)
proof (*intro conjI*)
from *Z-s-unroll noabort s-in-b* **show** *?C1* **by** *blast*
next
{
fix t
assume *s-t*: $\Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow\ Normal\ t$
moreover
from *Z-s-unroll s-t s-in-b*
have $(Z,\ t) \in \ ?unroll$
by (*blast intro*: *rtrancl-into-rtrancl*)
moreover note *noabort*
ultimately
have $(Z,\ t) \in \ ?unroll\ \wedge$
$(\forall\, e.\ (Z,e) \in ?unroll \longrightarrow e \in b$
$\longrightarrow \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ `\ (-F))\ \wedge$
$(\forall\, u.\ \Gamma \vdash \langle c, Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma \vdash \langle While\ b\ c, Normal\ Z\rangle \Rightarrow\ Abrupt\ u))$
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 $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
by $(\mathit{rule}\ \mathit{exec.WhileTrue})$
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 $(\mathit{Call}\ p)$
let $?P = \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
from *noStuck-Call* **have** $\forall s \in\ ?P.\ p \in \mathit{dom}\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show $\Gamma,\Theta\vdash_{/F}\ ?P\ (\mathit{Call}\ p)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule conseq-extract-state-indep-prop*)
assume *p-definied*: $p \in \mathit{dom}\ \Gamma$
with *MGT-Calls* **show**
$\Gamma,\Theta\vdash_{/F}\{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
$(\mathit{Call}\ p)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ \ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*auto*)
qed
next
case $(\mathit{DynCom}\ c)$
have *hyp*:
$\bigwedge s'.\ \forall Z.\ \Gamma,\Theta\vdash_{/F}\{s.\ s = Z \wedge \Gamma\vdash\langle c\ s',\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}$
$c\ s'$
$\{t.\ \Gamma\vdash\langle c\ s',\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},\{t.\ \Gamma\vdash\langle c\ s',\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta\vdash_{/F}\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{DynCom}\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))\}\ c$
Z
$\{t.\ \Gamma\vdash\langle \mathit{DynCom}\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},\{t.\ \Gamma\vdash\langle \mathit{DynCom}\ c,\mathit{Normal}\ Z\rangle$
$\Rightarrow \mathit{Abrupt}\ t\}$
by (*rule ConseqMGT* [*OF hyp*])
(*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
show $\Gamma,\Theta\vdash_{/F}\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{DynCom}\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F))\}$
$\mathit{DynCom}\ c$

```
            {t. Γ⊢⟨DynCom c,Normal Z⟩ ⇒ Normal t},
            {t. Γ⊢⟨DynCom c,Normal Z⟩ ⇒ Abrupt t}
    apply (rule hoarep.DynCom)
    apply (clarsimp)
    apply (rule hyp' [simplified])
    done
next
  case (Guard f g c)
  have hyp-c: ∀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}
    using Guard by iprover
  show ?case
  proof (cases f ∈ F)
    case True
    from hyp-c
    have Γ,Θ⊢/F (g ∩ {s. s = Z ∧
                 Γ⊢⟨Guard f g c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (− F))})
           c
          {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Normal t},
          {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Abrupt t}
      apply (rule ConseqMGT)
      apply (insert True)
      apply (auto simp add: final-notin-def intro: exec.intros)
      done
    from True this
    show ?thesis
      by (rule conseqPre [OF Guarantee]) auto
  next
    case False
    from hyp-c
    have Γ,Θ⊢/F
        (g ∩ {s. s=Z ∧ Γ⊢⟨Guard f g c,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))})

          c
          {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Normal t},
          {t. Γ⊢⟨Guard f g c,Normal Z⟩ ⇒ Abrupt t}
      apply (rule ConseqMGT)
      apply clarify
      apply (frule Guard-noFaultStuckD [OF - False])
      apply (auto simp add: final-notin-def intro: exec.intros)
      done
    then show ?thesis
      apply (rule conseqPre [OF hoarep.Guard])
      apply clarify
      apply (frule Guard-noFaultStuckD [OF - False])
      apply auto
      done
```

qed
next
case *Throw*
show $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$ *Throw*
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Throw},\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule conseqPre* [*OF hoarep.Throw*]) (*blast intro*: *exec.intros*)
next
case (*Catch* c_1 c_2)
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle c_1,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$ c_1
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$ c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\ \wedge$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{/F}$ $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$ c_2
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Catch.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{/F}\{s.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ s\ \wedge$
$\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
c_2
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule ConseqMGT*)
(*fastforce intro*: *exec.intros simp add*: *final-notin-def*)
ultimately
show $\Gamma,\Theta\vdash_{/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
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 hoarep.Catch*)
qed

lemma *MGT-Calls*:
$\forall p{\in}\mathit{dom}\ \Gamma.\ \forall Z.$
$\Gamma,\{\}\vdash_{/F}\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))\}$
$(\mathit{Call}\ p)$

```
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t}
proof –
 {
   fix p Z
   assume defined: p ∈ dom Γ
   have
    Γ,(⋃p∈dom Γ. ⋃Z.
        {({s. s=Z ∧
           Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))},
           p,
           {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
           {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t})})
     ⊢/F{s. s = Z ∧ Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
        (the (Γ p))
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
        {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t}
    (is Γ,?Θ ⊢/F (?Pre p Z) (the (Γ p)) (?Post p Z),(?Abr p Z))
   proof –
     have MGT-Calls:
      ∀p∈dom Γ. ∀Z. Γ,?Θ ⊢/F
       {s. s=Z ∧ Γ⊢⟨Call p,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))}
        (Call p)
       {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Normal t},
       {t. Γ⊢⟨Call p,Normal Z⟩ ⇒ Abrupt t}
       by (intro ballI allI, rule HoarePartialDef.Asm,auto)
       have ∀Z. Γ,?Θ⊢/F {s. s=Z ∧ Γ⊢⟨the (Γ p) ,Normal s⟩ ⇒∉({Stuck} ∪
Fault‘(−F))}
                     (the (Γ p))
                     {t. Γ⊢⟨the (Γ p),Normal Z⟩ ⇒ Normal t},
                     {t. Γ⊢⟨the (Γ p),Normal Z⟩ ⇒ Abrupt t}
       by (iprover intro: MGT-lemma [OF MGT-Calls])
     thus Γ,?Θ⊢/F (?Pre p Z) (the (Γ p)) (?Post p Z),(?Abr p Z)
       apply (rule ConseqMGT)
       apply (clarify,safe)
     proof –
       assume Γ⊢⟨Call p,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
       with defined show Γ⊢⟨the (Γ p),Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
         by (fastforce simp add: final-notin-def
               intro: exec.intros)
     next
       fix t
       assume Γ⊢⟨the (Γ p),Normal Z⟩ ⇒ Normal t
       with defined
       show Γ⊢⟨Call p,Normal Z⟩ ⇒Normal t
         by  (auto intro: exec.Call)
     next
       fix t
```

assume $\Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Abrupt\ t$
with *defined*
show $\Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*auto intro*: *exec.Call*)
qed
qed
}
then show *?thesis*
apply −
apply (*intro ballI allI*)
apply (*rule CallRec′* [**where** $Procs=dom\ \Gamma$ **and**
$P=\lambda p\ Z.\ \{s.\ s=Z \wedge$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$**and**
$Q=\lambda p\ Z.$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\}$ **and**
$A=\lambda p\ Z.$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$])
apply *simp+*
done
qed

theorem *hoare-complete*: $\Gamma\models_{/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\{\}\vdash_{/F}\ P\ c\ Q,A$
by (*iprover intro*: *MGT-implies-complete MGT-lemma* [*OF MGT-Calls*])

lemma *hoare-complete′*:
assumes *cvalid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
proof (*cases* $\Gamma\models_{/F}\ P\ c\ Q,A$)
case *True*
hence $\Gamma,\{\}\vdash_{/F}\ P\ c\ Q,A$
by (*rule hoare-complete*)
thus $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
by (*rule hoare-augment-context*) *simp*
next
case *False*
with *cvalid*
show *?thesis*
by (*rule ExFalso*)
qed

lemma *hoare-strip-*Γ:
assumes *deriv*: $\Gamma,\{\}\vdash_{/F}\ P\ p\ Q,A$
assumes F': $F' \subseteq -F$
shows *strip* $F'\ \Gamma,\{\}\vdash_{/F}\ P\ p\ Q,A$
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

8.3 And Now: Some Useful Rules

8.3.1 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
$procs\ Skip = \{\}\ |$
$procs\ (Basic\ f) = \{\}\ |$
$procs\ (Seq\ c_1\ c_2) = (procs\ c_1 \cup procs\ c_2)\ |$
$procs\ (Cond\ b\ c_1\ c_2) = (procs\ c_1 \cup procs\ c_2)\ |$
$procs\ (While\ b\ c) = procs\ c\ |$
$procs\ (Call\ p) = \{p\}\ |$
$procs\ (DynCom\ c) = (\bigcup s.\ procs\ (c\ s))\ |$
$procs\ (Guard\ f\ g\ c) = procs\ c\ |$
$procs\ Throw = \{\}\ |$
$procs\ (Catch\ c_1\ c_2) = (procs\ c_1 \cup procs\ c_2)$

primrec *noSpec*:: $('s,'p,'f)\ com \Rightarrow bool$
where

noSpec Skip = *True* |
noSpec (*Basic f*) = *True* |
noSpec (*Spec r*) = *False* |
noSpec (*Seq* c_1 c_2) = (*noSpec* c_1 ∧ *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*: ∀ *Z*. ∀ *n*. Γ,Θ⊨*n*:$_{/F}$ (*P′ Z*) *c* (*Q′ Z*),(*A′ Z*)
assumes *cons*: ∀ *s t*. (∀ *Z*. *s* ∈ *P′ Z* ⟶ *t* ∈ *Q′ Z*)
⟶ (*s* ∈ *P* ⟶ *t* ∈ *Q*)
assumes *noguards-c*: *noguards c*

assumes *noguards-*Γ: ∀ *p* ∈ *dom* Γ. *noguards* (*the* (Γ *p*))
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: ∀ *p* ∈ *dom* Γ. *nothrows* (*the* (Γ *p*))
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-*Γ: ∀ *p* ∈ *dom* Γ. *noSpec* (*the* (Γ *p*))
assumes *procs-subset*: *procs c* ⊆ *dom* Γ
assumes *procs-subset-*Γ: ∀ *p* ∈ *dom* Γ. *procs* (*the* (Γ *p*)) ⊆ *dom* Γ
shows Γ,Θ⊨*n*:$_{/F}$ *P c Q*,*A*
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*:∀ (*P*, *p*, *Q*, *A*)∈Θ. Γ⊨*n*:$_{/F}$ *P* (*Call p*) *Q*,*A*
  assume *exec*: Γ⊢⟨*c*,*Normal s*⟩ =*n*⇒ *t*
  assume *P*: *s* ∈ *P*
  assume *t-notin-F*: *t* ∉ *Fault* ‘ *F*
  show *t* ∈ *Normal* ‘ *Q* ∪ *Abrupt* ‘ *A*
  proof −
    from *execn-noguards-no-Fault* [*OF exec noguards-c noguards-*Γ]
     execn-nothrows-no-Abrupt [*OF exec nothrows-c nothrows-*Γ]
     execn-noSpec-no-Stuck [*OF exec*
          *noSpec-c noSpec-*Γ *procs-subset*
      *procs-subset-*Γ]
    obtain *t′* **where** *t*: *t*=*Normal t′*
      by (*cases t*) *auto*
    with *exec spec ctxt*
    have (∀ *Z*. *s* ∈ *P′ Z* ⟶ *t′* ∈ *Q′ Z*)
      by (*unfold cnvalid-def nvalid-def*) *blast*
    with *cons P t* **show** *?thesis*
      by *simp*
  qed
qed

lemma *LiberalConseq-noguards-nothrows*:
assumes *spec*: ∀ *Z*. Γ,Θ⊢$_{/F}$ (*P′ Z*) *c* (*Q′ Z*),(*A′ Z*)
assumes *cons*: ∀ *s t*. (∀ *Z*. *s* ∈ *P′ Z* ⟶ *t* ∈ *Q′ Z*)
          ⟶ (*s* ∈ *P* ⟶ *t* ∈ *Q*)
assumes *noguards-c*: *noguards c*
assumes *noguards-*Γ: ∀ *p* ∈ *dom* Γ. *noguards* (*the* (Γ *p*))
assumes *nothrows-c*: *nothrows c*
assumes *nothrows-*Γ: ∀ *p* ∈ *dom* Γ. *nothrows* (*the* (Γ *p*))
assumes *noSpec-c*: *noSpec c*
assumes *noSpec-*Γ: ∀ *p* ∈ *dom* Γ. *noSpec* (*the* (Γ *p*))
assumes *procs-subset*: *procs c* ⊆ *dom* Γ
assumes *procs-subset-*Γ: ∀ *p* ∈ *dom* Γ. *procs* (*the* (Γ *p*)) ⊆ *dom* Γ
shows Γ,Θ⊢$_{/F}$ *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

8.3.2 Modify Return

lemma *ProcModifyReturn-sound*:
assumes *valid-call*: $\forall n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ call\ init\ p\ return'\ c\ Q,A$
assumes *valid-modif*:
$\forall \sigma.\ \forall n.\ \Gamma,\Theta\models n{:}_{/UNIV}\ \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
assumes *ret-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s)$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s)$
$\longrightarrow return'\ s\ t = return\ s\ t$
shows $\Gamma,\Theta \models n{:}_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
then have *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/UNIV}\ P\ (Call\ p)\ Q,A$
by (*auto intro*: *nvalid-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle call\ init\ p\ return\ c,Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault\ `\ F$
from *exec*

show *t* ∈ *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 Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ =*m*⇒ *Fault f n* = *Suc m*
    t = *Fault f*
  with *bdy* **have** Γ⊢⟨*call init p return′ c* ,*Normal s*⟩ =*n*⇒ *t*

by (*auto intro*: *execn-callFault*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
next
fix *bdy m*
assume *bdy*: $\Gamma\ p = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow Stuck\ n = Suc\ m$
$t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
by (*auto intro*: *execn-callStuck*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
next
fix *m*
assume $\Gamma\ p = None$
and $n = Suc\ m\ t = Stuck$
then have $\Gamma\vdash\langle call\ init\ p\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow t$
by (*auto intro*: *execn-callUndefined*)
from *valid-call* [*rule-format*] *ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
qed
qed

lemma *ProcModifyReturn*:
assumes *spec*: $\Gamma,\Theta\vdash_{/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
assumes *result-conform*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *return-conform*:
$\forall s\ t.\ t \in ModifAbr\ (init\ s)$
$\longrightarrow (return'\ s\ t) = (return\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ Call\ p\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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*: Γ *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*: 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 *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*:

$\forall s\ t.\ t \in \mathit{ModifAbr}\ (\mathit{init}\ s) \longrightarrow (\mathit{return}'\ s\ t) = (\mathit{return}\ s\ t)$
assumes *modifies-spec*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F}\ \{\sigma\}\ \mathit{Call}\ p\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (\mathit{call\ init\ p\ return\ c})\ Q,A$
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

8.3.3 DynCall

lemma *dynProcModifyReturn-sound*:
assumes *valid-call*: $\bigwedge n.\ \Gamma,\Theta \models n{:}_{/F}\ P\ \mathit{dynCall\ init\ p\ return'\ c}\ Q,A$
assumes *valid-modif*:
$\forall s \in P.\ \forall \sigma.\ \forall n.$
$\Gamma,\Theta\models n{:}_{/UNIV}\ \{\sigma\}\ \mathit{Call}\ (p\ s)\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
assumes *ret-modif*:
$\forall s\ t.\ t \in \mathit{Modif}\ (\mathit{init}\ s)$
$\longrightarrow \mathit{return}'\ s\ t = \mathit{return}\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in \mathit{ModifAbr}\ (\mathit{init}\ s)$
$\longrightarrow \mathit{return}'\ s\ t = \mathit{return}\ s\ t$
shows $\Gamma,\Theta \models n{:}_{/F}\ P\ (\mathit{dynCall\ init\ p\ return\ c})\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/F}\ P\ (\mathit{Call}\ p)\ Q,A$
then have *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models n{:}_{/UNIV}\ P\ (\mathit{Call}\ p)\ Q,A$
by (*auto intro*: *nvalid-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle \mathit{dynCall\ init\ p\ return\ c},\mathit{Normal}\ s\rangle =n\Rightarrow t$
assume *t-notin-F*: $t \notin \mathit{Fault}$ ‘ F
assume *P*: $s \in P$
with *valid-modif*
have *valid-modif'*: $\forall \sigma.\ \forall n.$
$\Gamma,\Theta\models n{:}_{/UNIV}\ \{\sigma\}\ \mathit{Call}\ (p\ s)\ (\mathit{Modif}\ \sigma),(\mathit{ModifAbr}\ \sigma)$
by *blast*
from *exec*
have $\Gamma\vdash\langle \mathit{call\ init}\ (p\ s)\ \mathit{return\ c},\mathit{Normal}\ s\rangle =n\Rightarrow t$
by (*cases rule*: *execn-dynCall-Normal-elim*)
then show $t \in \mathit{Normal}$ ‘ $Q \cup \mathit{Abrupt}$ ‘ A
proof (*cases rule*: *execn-call-Normal-elim*)
fix *bdy m t'*
assume *bdy*: $\Gamma\ (p\ s) = \mathit{Some\ bdy}$
assume *exec-body*: $\Gamma\vdash\langle \mathit{bdy},\mathit{Normal}\ (\mathit{init}\ s)\rangle =m\Rightarrow \mathit{Normal}\ t'$

assume *exec-c*: $\Gamma\vdash\langle c\ s\ t',Normal\ (return\ s\ t')\rangle =Suc\ m\Rightarrow\ t$
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*: $\Gamma\ (p\ s) = Some\ bdy$
assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow\ Abrupt\ t'$
assume *n*: $n = Suc\ m$
assume *t*: $t = Abrupt\ (return\ s\ t')$
also from *exec-body n bdy*
have $\Gamma\vdash\langle Call\ (p\ s)\ ,Normal\ (init\ s)\rangle =n\Rightarrow\ Abrupt\ t'$
  by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt' this*] *P*
have $t' \in ModifAbr\ (init\ s)$
  by *auto*
with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
  by *simp*
finally have $t = Abrupt\ (return'\ s\ t')$ **.**
with *exec-body bdy n*
have $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c,Normal\ s\rangle =n\Rightarrow\ t$
  by (*auto intro*: *execn-callAbrupt*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow\ t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m f*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle =m\Rightarrow\ Fault\ f\ n = Suc\ m$
  $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle =n\Rightarrow\ t$
  by (*auto intro*: *execn-callFault*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle =n\Rightarrow\ t$

    by (*rule execn-dynCall*)
   from *valid-call ctxt this P t-notin-F*
   show *?thesis*
    by (*rule cnvalidD*)
  next
   fix *bdy m*
   assume *bdy*: Γ (*p s*) = *Some bdy*
   assume $\Gamma\vdash\langle$*bdy*,*Normal* (*init s*)$\rangle$ =*m*$\Rightarrow$ *Stuck* *n* = *Suc m*
    t = *Stuck*
   with *bdy* **have** $\Gamma\vdash\langle$*call init* (*p s*) *return′ c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
    by (*auto intro*: *execn-callStuck*)
   hence $\Gamma\vdash\langle$*dynCall init p return′ c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
    by (*rule execn-dynCall*)
   from *valid-call ctxt this P t-notin-F*
   show *?thesis*
    by (*rule cnvalidD*)
  next
   fix *m*
   assume Γ (*p s*) = *None*
   and *n* = *Suc m t* = *Stuck*
   hence $\Gamma\vdash\langle$*call init* (*p s*) *return′ c* ,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
    by (*auto intro*: *execn-callUndefined*)
   hence $\Gamma\vdash\langle$*dynCall init p return′ c*,*Normal s*$\rangle$ =*n*$\Rightarrow$ *t*
    by (*rule execn-dynCall*)
   from *valid-call ctxt this P t-notin-F*
   show *?thesis*
    by (*rule cnvalidD*)
  qed
qed

lemma *dynProcModifyReturn*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{/F}$ *P dynCall init p return′ c Q*,*A*
assumes *ret-modif*:
  $\forall$ *s t*. *t* $\in$ *Modif* (*init s*)
     $\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall$ *s t*. *t* $\in$ *ModifAbr* (*init s*)
               $\longrightarrow$ *return′ s t* = *return s t*
assumes *modif*:
  $\forall$ *s* $\in$ *P*. $\forall\sigma$.
   $\Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ *Call* (*p s*) (*Modif* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{/F}$ *P* (*dynCall init p return c*) *Q*,*A*
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule dynProcModifyReturn-sound* [**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
    OF hoare-cnvalid [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-cnvalid* [*OF modif* [*rule-format*]])
apply *assumption*
**done*

lemma *dynProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\bigwedge n.\ \Gamma,\Theta \models n:_{/F}\ P\ dynCall\ init\ p\ return'\ c\ Q,A$
assumes *valid-modif*:
  $\forall s \in P.\ \forall \sigma.\ \forall n.$
    $\Gamma,\Theta{\models}n:_{/F}\ \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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*: $\Gamma\ (p\ s) = Some\ bdy$

assume *exec-body*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ =m\Rightarrow\ Abrupt\ t'$
assume *n*: $n = Suc\ m$
assume *t*: $t = Abrupt\ (return\ s\ t')$
also from *exec-body n bdy*
have $\Gamma\vdash\langle Call\ (p\ s)\ ,Normal\ (init\ s)\rangle\ =n\ \Rightarrow\ Abrupt\ t'$
  by (*auto simp add*: *intro*: *execn.intros*)
from *cnvalidD* [*OF valid-modif'* [*rule-format*, *of n init s*] *ctxt this*] *P*
have $t' \in ModifAbr\ (init\ s)$
  by *auto*
with *ret-modifAbr* **have** $Abrupt\ (return\ s\ t') = Abrupt\ (return'\ s\ t')$
  by *simp*
finally have $t = Abrupt\ (return'\ s\ t')$ **.**
with *exec-body bdy n*
have $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*auto intro*: *execn-callAbrupt*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m f*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ =m\Rightarrow\ Fault\ f\ n = Suc\ m$ **and**
  t: $t = Fault\ f$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*auto intro*: *execn-callFault*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*rule execn-dynCall*)
from *cnvalidD* [*OF valid-call ctxt this P*] *t t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy m*
assume *bdy*: $\Gamma\ (p\ s) = Some\ bdy$
assume $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle\ =m\Rightarrow\ Stuck\ n = Suc\ m$
  $t = Stuck$
with *bdy* **have** $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*auto intro*: *execn-callStuck*)
hence $\Gamma\vdash\langle dynCall\ init\ p\ return'\ c,Normal\ s\rangle\ =n\Rightarrow\ t$
  by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
  by (*rule cnvalidD*)
next
fix *m*
assume $\Gamma\ (p\ s) = None$
and $n = Suc\ m\ t = Stuck$
hence $\Gamma\vdash\langle call\ init\ (p\ s)\ return'\ c\ ,Normal\ s\rangle\ =n\Rightarrow\ t$

by (*auto intro*: *execn-callUndefined*)
hence $\Gamma\vdash\langle$*dynCall init p return′ c*,*Normal s*$\rangle$ $=n\Rightarrow$ *t*
by (*rule execn-dynCall*)
from *valid-call ctxt this P t-notin-F*
show *?thesis*
by (*rule cnvalidD*)
qed
qed

lemma *dynProcModifyReturnSameFaults*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{/F}$ *P dynCall init p return′ c Q*,*A*
assumes *ret-modif*:
$\forall s\ t.\ t \in$ *Modif* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
assumes *ret-modifAbr*: $\forall s\ t.\ t \in$ *ModifAbr* (*init s*)
$\longrightarrow$ *return′ s t* = *return s t*
assumes *modif*:
$\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}$ *Call* (*p s*) (*Modif* 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

8.3.4 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

8.3.5 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

8.3.6 Guards and Guarantees

lemma *SplitGuards-sound*:
assumes *valid-c1*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c_1\ Q,A$
assumes *valid-c2*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c_2\ UNIV,UNIV$

assumes *c*: $(c_1 \cap_g c_2) = Some\ c$
shows $\Gamma,\Theta \models n{:}_{/F}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma \vdash \langle c, Normal\ s\rangle =n\Rightarrow t$
  assume *P*: $s \in P$
  assume *t-notin-F*: $t \notin Fault\ `\ F$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases t*)
    case *Normal*
    with *inter-guards-execn-noFault* [*OF c exec*]
    have $\Gamma \vdash \langle c_1, Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
    from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
    show *?thesis*
      by (*rule cnvalidD*)
  next
    case *Abrupt*
    with *inter-guards-execn-noFault* [*OF c exec*]
    have $\Gamma \vdash \langle c_1, Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
    from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
    show *?thesis*
      by (*rule cnvalidD*)
  next
    case (*Fault f*)
    with *exec inter-guards-execn-Fault* [*OF c*]
    have $\Gamma \vdash \langle c_1, Normal\ s\rangle =n\Rightarrow Fault\ f \vee \Gamma \vdash \langle c_2, Normal\ s\rangle =n\Rightarrow Fault\ f$
      by *auto*
    then show *?thesis*
    proof (*cases rule*: *disjE* [*consumes 1*])
      assume $\Gamma \vdash \langle c_1, Normal\ s\rangle =n\Rightarrow Fault\ f$
      from *Fault cnvalidD* [*OF valid-c1* [*rule-format*] *ctxt this P*] *t-notin-F*
      show *?thesis*
        by *blast*
    next
      assume $\Gamma \vdash \langle c_2, Normal\ s\rangle =n\Rightarrow Fault\ f$
      from *Fault cnvalidD* [*OF valid-c2* [*rule-format*] *ctxt this P*] *t-notin-F*
      show *?thesis*
        by *blast*
    qed
  next
    case *Stuck*
    with *inter-guards-execn-noFault* [*OF c exec*]
    have $\Gamma \vdash \langle c_1, Normal\ s\rangle =n\Rightarrow t$ **by** *simp*
    from *valid-c1* [*rule-format*] *ctxt this P t-notin-F*
    show *?thesis*
      by (*rule cnvalidD*)
  qed
qed

lemma *SplitGuards*:
  assumes *c*: $(c_1 \cap_g c_2) = Some\ c$
  assumes *deriv-c1*: $\Gamma,\Theta\vdash_{/F}\ P\ c_1\ Q,A$
  assumes *deriv-c2*: $\Gamma,\Theta\vdash_{/F}\ P\ c_2\ UNIV,UNIV$
  shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule SplitGuards-sound* [*OF - - c*])
using *deriv-c1*
apply (*blast intro*: *hoare-cnvalid*)
using *deriv-c2*
apply (*blast intro*: *hoare-cnvalid*)
done

lemma *CombineStrip-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
  assumes *valid-strip*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}\ P\ (strip\text{-}guards\ (-F)\ c)\ UNIV,UNIV$
  shows $\Gamma,\Theta\models n{:}_{/\{\}}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}\ P\ (Call\ p)\ Q,A$
  hence *ctxt'*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle\ =n\Rightarrow\ t$
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin Fault\ `\ \{\}$
  show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$
  proof (*cases t*)
    case (*Normal t'*)
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Normal*
    show *?thesis*
      by *auto*
  next
    case (*Abrupt t'*)
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Abrupt*
    show *?thesis*
      by *auto*
  next
    case (*Fault f*)
    show *?thesis*
    proof (*cases* $f \in F$)
      case *True*
      hence $f \notin -F$ **by** *simp*
      with *exec Fault*
      have $\Gamma\vdash\langle strip\text{-}guards\ (-F)\ c,Normal\ s\rangle\ =n\Rightarrow\ Fault\ f$
        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*: $\Gamma,\Theta\vdash_{/\{\}}\ P\ (\textit{strip-guards}\ (-F)\ c)\ \textit{UNIV},\textit{UNIV}$
  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*: $\forall n.\ \Gamma,\Theta\models n{:}_{/-F}\ P\ c\ \textit{UNIV},\textit{UNIV}$
  shows $\Gamma,\Theta\models n{:}_{/\{\}}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}\ P\ (\textit{Call}\ p)\ Q,A$
  hence *ctxt′*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (\textit{Call}\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  from *ctxt* **have** *ctxtFlip*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/-F}\ P\ (\textit{Call}\ p)\ Q,A$
    by (*auto intro*: *nvalid-augment-Faults*)
  assume *exec*: $\Gamma\vdash\langle c,\textit{Normal}\ s\rangle\ =n\Rightarrow\ t$
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin \textit{Fault}\ `\ \{\}$
  show $t \in \textit{Normal}\ `\ Q \cup \textit{Abrupt}\ `\ A$
  proof (*cases t*)
    case (*Normal t′*)
    from *cnvalidD* [*OF valid* [*rule-format*] *ctxt′ exec P*] *Normal*
    show *?thesis*
      by *auto*

next
  case (*Abrupt* t')
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Abrupt*
  show *?thesis*
    by *auto*
next
  case (*Fault f*)
  show *?thesis*
  proof (*cases* $f \in F$)
    case *True*
    hence $f \notin -F$ **by** *simp*
    with *cnvalidD* [*OF validFlip* [*rule-format*] *ctxtFlip exec P*] *Fault*
    have *False*
      by *auto*
    thus *?thesis* **..**
  next
    case *False*
    with *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Fault*
    show *?thesis*
      by *auto*
  qed
next
  case *Stuck*
  from *cnvalidD* [*OF valid* [*rule-format*] *ctxt' exec P*] *Stuck*
  show *?thesis*
    by *auto*
qed
qed

lemma *GuardsFlip*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ $P\ c\ Q,A$
  assumes *derivFlip*: $\Gamma,\Theta\vdash_{/-F}$ $P\ c\ UNIV,UNIV$
  shows $\Gamma,\Theta\vdash_{/\{\}}$ $P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule GuardsFlip-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
apply (*iprover intro*: *hoare-cnvalid* [*OF derivFlip*])
done

lemma *MarkGuardsI-sound*:
  assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}$ $P\ c\ Q,A$
  shows $\Gamma,\Theta\models n{:}_{/\{\}}$ P *mark-guards f c* Q,A
proof (*rule cnvalidI*)
  fix $s\ t$
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}\ P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle$*mark-guards f c*,*Normal s*$\rangle$ $=n\Rightarrow$ t
  from *execn-mark-guards-to-execn* [*OF exec*] **obtain** t' **where**

exec-c: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t'$ **and**
t'-noFault: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ` $\{\}$
show $t \in$ *Normal* ` $Q \cup$ *Abrupt* ` A
proof $-$
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-c P*]
have $t' \in$ *Normal* ` $Q \cup$ *Abrupt* ` A
by *blast*
with *t'-noFault*
show *?thesis*
by *auto*
qed
qed

lemma *MarkGuardsI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/\{\}}\ P\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{/\{\}}\ P$ *mark-guards f c* Q,A
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule MarkGuardsI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MarkGuardsD-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}\ P$ *mark-guards f c* Q,A
shows $\Gamma,\Theta\models n{:}_{/\{\}}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ` $\{\}$
show $t \in$ *Normal* ` $Q \cup$ *Abrupt* ` A
proof (*cases isFault t*)
case *True*
with *execn-to-execn-mark-guards-Fault* [*OF exec*]
obtain f' **where** $\Gamma\vdash\langle$*mark-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*: $\Gamma\vdash\langle merge\text{-}guards\ c,Normal\ s\rangle =n\Rightarrow t$
from *execn-merge-guards-to-execn* [*OF exec-merge*]
have *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$ **.**
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin Fault$ ‘ F
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec P t-notin-F*]
show $t \in Normal$ ‘ $Q \cup Abrupt$ ‘ A**.**
qed

lemma *MergeGuardsI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ *P c Q*,*A*
shows $\Gamma,\Theta\vdash_{/F}$ *P merge-guards c Q*,*A*
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule MergeGuardsI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *MergeGuardsD-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}$ *P merge-guards c Q*,*A*
shows $\Gamma,\Theta\models n{:}_{/F}$ *P c Q*,*A*
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
from *execn-to-execn-merge-guards* [*OF exec*]

have *exec-merge*: $\Gamma\vdash\langle$*merge-guards c*,*Normal s*$\rangle$ $=n\Rightarrow$ t**.**
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-merge P t-notin-F*]
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ *A***.**
qed

lemma *MergeGuardsD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}$ *P merge-guards c Q*,*A*
shows $\Gamma,\Theta\vdash_{/F}$ *P c Q*,*A*
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule MergeGuardsD-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *SubsetGuards-sound*:
assumes *c-c′*: $c \subseteq_g c'$
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/\{\}}$ *P c′ Q*,*A*
shows $\Gamma,\Theta\models n{:}_{/\{\}}$ *P c Q*,*A*
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models n{:}_{/\{\}}$ *P* (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle c$,*Normal s*$\rangle$ $=n\Rightarrow$ t
from *execn-to-execn-subseteq-guards* [*OF c-c′ exec*] **obtain** t' **where**
exec-c′: $\Gamma\vdash\langle c'$,*Normal s*$\rangle$ $=n\Rightarrow$ t' **and**
t'*-noFault*: $\neg$ *isFault* $t' \longrightarrow t' = t$
by *blast*
assume *P*: $s \in P$
assume *t-noFault*: $t \notin$ *Fault* ‘ $\{\}$
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-c′ P*] t'*-noFault t-noFault*
show $t \in$ *Normal* ‘ $Q \cup$ *Abrupt* ‘ *A*
by *auto*
qed

lemma *SubsetGuards*:
assumes *c-c′*: $c \subseteq_g c'$
assumes *deriv*: $\Gamma,\Theta\vdash_{/\{\}}$ *P c′ Q*,*A*
shows $\Gamma,\Theta\vdash_{/\{\}}$ *P c Q*,*A*
apply (*rule hoare-complete′*)
apply (*rule allI*)
apply (*rule SubsetGuards-sound* [*OF c-c′*])
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *NormalizeD-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}$ *P* (*normalize c*) *Q*,*A*

shows $\Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
hence *exec-norm*: $\Gamma\vdash\langle normalize\ c,Normal\ s\rangle =n\Rightarrow t$
by (*rule execn-to-execn-normalize*)
assume *P*: $s \in P$
assume *noFault*: $t \notin Fault\ `\ F$
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec-norm P noFault*]
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$.
qed

lemma *NormalizeD*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ (normalize\ c)\ Q,A$
shows $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule NormalizeD-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

lemma *NormalizeI-sound*:
assumes *valid*: $\forall n.\ \Gamma,\Theta\models n{:}_{/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\models n{:}_{/F}\ P\ (normalize\ c)\ Q,A$
proof (*rule cnvalidI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A){\in}\Theta.\ \Gamma\models n{:}_{/F}\ P\ (Call\ p)\ Q,A$
assume $\Gamma\vdash\langle normalize\ c,Normal\ s\rangle =n\Rightarrow t$
hence *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle =n\Rightarrow t$
by (*rule execn-normalize-to-execn*)
assume *P*: $s \in P$
assume *noFault*: $t \notin Fault\ `\ F$
from *cnvalidD* [*OF valid* [*rule-format*] *ctxt exec P noFault*]
show $t \in Normal\ `\ Q \cup Abrupt\ `\ A$.
qed

lemma *NormalizeI*:
assumes *deriv*: $\Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (normalize\ c)\ Q,A$
apply (*rule hoare-complete'*)
apply (*rule allI*)
apply (*rule NormalizeI-sound*)
apply (*iprover intro*: *hoare-cnvalid* [*OF deriv*])
done

8.3.7 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 Γ,{}$\vdash_{/F}$ 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*: Γ,{}$\vdash_{/F}$ P c Q,A
assumes F: $F \subseteq F'$
shows Γ,{}$\vdash_{/F'}$ 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

9 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*:$[\![\Gamma,\Theta\vdash_{/F} P'\ c\ Q',A';$
$\quad\forall s \in P.\ s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
apply (*rule conseq*)
apply (*rule allI*)
apply *assumption*
apply *auto*
done

lemma *conseq-Kleymann*:$[\![\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ c\ (Q'\ Z),(A'\ Z);$
$\quad\forall s \in P.\ (\exists Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))]\!]$
$\quad\Longrightarrow$
$\quad\Gamma,\Theta\vdash_{/F} P\ c\ Q,A$
by (*rule Conseq'*) *blast*

lemma *DynComConseq*:
assumes $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \ \Gamma,\Theta\vdash_{/F} P'\ (c\ s)\ Q',A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
shows $\Gamma,\Theta\vdash_{/F} P$ *DynCom* $c\ Q,A$
using *assms*
apply −
apply (*rule DynCom*)
apply *clarsimp*
apply (*rule Conseq*)
apply *clarsimp*
apply *blast*
done

lemma *SpecAnno*:
assumes *consequence*: $P \subseteq \{s.\ (\exists\ Z.\ s{\in}P'\ Z \wedge (Q'\ Z \subseteq Q) \wedge (A'\ Z \subseteq A))\}$
assumes *spec*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),(A'\ Z)$
assumes *bdy-constant*: $\forall Z.\ c\ Z = c$ *undefined*
shows $\Gamma,\Theta\vdash_{/F} P$ (*specAnno* $P'\ c\ Q'\ A'$) Q,A
proof −
from *spec bdy-constant*
have $\forall Z.\ \Gamma,\Theta\vdash_{/F} ((P'\ Z))\ (c$ *undefined*$)\ (Q'\ Z),(A'\ Z)$
apply −
apply (*rule allI*)
apply (*erule-tac x=Z* **in** *allE*)
apply (*erule-tac x=Z* **in** *allE*)
apply *simp*
done
with *consequence* **show** *?thesis*
apply (*simp add*: *specAnno-def*)
apply (*erule conseq*)

apply *blast*
done
qed

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

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

lemma *Skip*: $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ \mathit{Skip}\ Q,A$
by (*rule hoarep.Skip* [*THEN conseqPre*],*simp*)

lemma *Basic*: $P \subseteq \{s.\ (f\ s) \in Q\} \Longrightarrow\ \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Basic}\ f)\ Q,A$
by (*rule hoarep.Basic* [*THEN conseqPre*])

lemma *BasicCond*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s{\in}Q) \wedge (\neg\ b\ s \longrightarrow g\ s{\in}Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ \mathit{Basic}\ (\lambda s.\ \mathit{if}\ b\ s\ \mathit{then}\ f\ s\ \mathit{else}\ g\ s)\ Q,A$
apply (*rule Basic*)
apply *auto*
done

lemma *Spec*: $P \subseteq \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Spec}\ r)\ Q,A$
by (*rule hoarep.Spec* [*THEN conseqPre*])

lemma *SpecIf*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{/F}\ P\ \mathit{Spec}\ (\mathit{if\text{-}rel}\ b\ f\ g\ h)\ Q,A$

apply (*rule Spec*)
apply (*auto simp add*: *if-rel-def*)
done

lemma *Seq* [*trans*, *intro?*]:
$[\![\Gamma,\Theta\vdash_{/F} P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P\ (Seq\ c_1\ c_2)\ Q,A$
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*:
$[\![\ \Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A;\ P \subseteq g \cap R]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
by (*rule Guard*)

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

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

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

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

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

lemma *GuaranteeStripSwap*:
$[\![\Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A;\ P \subseteq R;\ f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{guaranteeStrip}\ f\ g\ c)\ Q,A$
by (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

lemma *GuaranteeAsGuard*:
$[\![P \subseteq g \cap R;\ \Gamma,\Theta\vdash_{/F}\ R\ c\ Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (\mathit{guaranteeStrip}\ f\ g\ c)\ Q,A$

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 ⊆ I
  assumes deriv-body: Γ,Θ⊢/F (I ∩ b) c I,A
  assumes I-Q: I ∩ −b ⊆ Q
  shows Γ,Θ⊢/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 ⊆ {s. (∃ Z. s∈I Z ∧ (I Z ∩ −b ⊆ Q)) }
assumes bdy: ∀ Z. Γ,Θ⊢/F (I Z ∩ b) (c Z) (I Z),A
assumes bdy-constant:  ∀ Z. c Z = c undefined
shows Γ,Θ⊢/F P (whileAnnoFix b I V c) Q,A
proof −
  from bdy bdy-constant
  have bdy': ∀ Z. Γ,Θ⊢/F (I Z ∩ 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 ∀ Z. Γ,Θ⊢/F (I Z) (whileAnnoFix b I V c) (I Z ∩ −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 ⊆ {s. (∃ Z. s∈I Z ∧
                               (∀ t. t ∈ I Z ∩ −b ⟶ t ∈ Q)) }
assumes bdy: ∀ Z. Γ,Θ⊢/F (I Z ∩ b) (c Z) (I Z),A
assumes bdy-constant:  ∀ Z. c Z = c undefined
shows Γ,Θ⊢/F P (whileAnnoFix b I V c) Q,A
  apply (rule WhileAnnoFix [OF - bdy bdy-constant])
  using consequence by blast

lemma WhileAnnoGFix:
```

assumes *whileAnnoFix*:
$\Gamma,\Theta\vdash_{/F}$ P (*guards gs*
(*whileAnnoFix* b J V (λZ. (*Seq* (c Z) (*guards gs Skip*))))) Q,A
shows $\Gamma,\Theta\vdash_{/F}$ P (*whileAnnoGFix gs b I V c*) Q,A
using *whileAnnoFix*
by (*simp add*: *whileAnnoGFix-def whileAnnoFix-def while-def*)

lemma *Bind*:
assumes *adapt*: $P \subseteq \{s.\ s \in P'\ s\}$
assumes *c*: $\forall s.\ \Gamma,\Theta\vdash_{/F} (P'\ s)\ (c\ (e\ s))\ Q,A$
shows $\Gamma,\Theta\vdash_{/F} P\ (bind\ e\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge s \in P'\ Z\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
apply (*rule allI*)
apply (*unfold bind-def*)
apply (*rule DynCom*)
apply (*rule ballI*)
apply *simp*
apply (*rule conseqPre*)
apply (*rule c* [*rule-format*])
apply *blast*
using *adapt*
apply *blast*
done

lemma *Block*:
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{/F} (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
shows $\Gamma,\Theta\vdash_{/F} P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge init\ s \in P'\ Z\}$ **and** $Q'=\lambda Z.\ Q$
and
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in R\ Z\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ Z\ t \in R\ Z\ t\}$ **and**
$Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule DynCom*)
apply (*clarsimp*)

apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ Z\ t \in A\}$ **in** *Catch*)
apply (*rule-tac* $R=\{i.\ i \in P'\ Z\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule bdy* [*rule-format*])
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *BlockSwap*:
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{/F} (P'\ s)\ bdy\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *adapt*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
shows $\Gamma,\Theta\vdash_{/F} P\ (block\ init\ bdy\ return\ c)\ Q,A$
using *adapt bdy c*
  by (*rule Block*)

lemma *BlockSpec*:
  assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
  assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
  assumes *bdy*: $\forall Z.\ \Gamma,\Theta\vdash_{/F} (P'\ Z)\ bdy\ (Q'\ Z),(A'\ Z)$
  shows $\Gamma,\Theta\vdash_{/F} P\ (block\ init\ bdy\ return\ c)\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ init\ s \in P'\ Z \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$ **and** $Q'=\lambda Z.\ Q$ **and**
$A'=\lambda Z.\ A$])
prefer *2*
using *adapt*
apply *blast*
apply (*rule allI*)
apply (*unfold block-def*)
apply (*rule DynCom*)
apply (*rule ballI*)
apply *clarsimp*
apply (*rule-tac* $R=\{t.\ return\ s\ t \in R\ s\ t\}$ **in** *SeqSwap*)
apply (*rule-tac* $P'=\lambda Z'.\ \{t.\ t=Z' \wedge return\ s\ t \in R\ s\ t\}$ **and**
$Q'=\lambda Z'.\ Q$ **and** $A'=\lambda Z'.\ A$ **in** *conseq*)

prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule DynCom*)
apply (*clarsimp*)
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
apply (*rule-tac R*=$\{t.$ *return s t* $\in A\}$ **in** *Catch*)
apply (*rule-tac R*=$\{i.\ i \in P'\ Z\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule conseq* [*OF bdy*])
apply *clarsimp*
apply *blast*
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule Basic*)
apply *simp*
done

lemma *Throw*: $P \subseteq A \Longrightarrow \Gamma,\Theta\vdash_{/F} P$ *Throw* Q,A
by (*rule hoarep.Throw* [*THEN conseqPre*])

lemmas *Catch* = *hoarep.Catch*
lemma *CatchSwap*: $[\![\Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A;\ \Gamma,\Theta\vdash_{/F} P\ c_1\ Q,R]\!] \Longrightarrow \Gamma,\Theta\vdash_{/F} P$ *Catch* $c_1\ c_2\ Q,A$
by (*rule hoarep.Catch*)

lemma *raise*: $P \subseteq \{s.\ f\ s \in A\} \Longrightarrow \Gamma,\Theta\vdash_{/F} P$ *raise f* Q,A
apply (*simp add*: *raise-def*)
apply (*rule Seq*)
apply (*rule Basic*)
apply (*assumption*)
apply (*rule Throw*)
apply (*rule subset-refl*)
done

lemma *condCatch*: $[\![\Gamma,\Theta\vdash_{/F} P\ c_1\ Q,((b \cap R) \cup (-b \cap A));\Gamma,\Theta\vdash_{/F} R\ c_2\ Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{/F}P$ *condCatch* $c_1\ b\ c_2\ Q,A$
apply (*simp add*: *condCatch-def*)
apply (*rule Catch*)
apply *assumption*
apply (*rule CondSwap*)
apply (*assumption*)
apply (*rule hoarep.Throw*)
apply *blast*

done

lemma *condCatchSwap*: $[\![\Gamma,\Theta\vdash_{/F}\ R\ c_2\ \ Q,A;\Gamma,\Theta\vdash_{/F}\ P\ c_1\ \ Q,((b\ \cap\ R)\ \cup\ (-b\ \cap\ A))]\!]$
$\Longrightarrow\ \Gamma,\Theta\vdash_{/F} P\ condCatch\ c_1\ b\ c_2\ Q,A$
by (*rule condCatch*)

lemma *ProcSpec*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t)\ \wedge$
$(\forall t.\ t \in A'\ Z \longrightarrow return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
using *adapt c p*
apply (*unfold call-def*)
by (*rule BlockSpec*)

lemma *ProcSpec′*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall t \in Q'\ Z.\ return\ s\ t \in R\ s\ t)\ \wedge$
$(\forall t \in A'\ Z.\ return\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{/F}\ (P'\ Z)\ Call\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{/F}\ P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
apply (*insert adapt*)
apply *clarsimp*
apply (*drule* (*1*) *subsetD*)
apply (*clarsimp*)
apply (*rule-tac x=Z* **in** *exI*)
apply *blast*
done

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

lemma *FCall*:

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

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

9.1 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*:

$\llbracket\Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ I\ V\ c)\ Q,A\rrbracket \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ (whileAnno\ b\ J\ V\ c)\ Q,A$
by (*simp add*: *whileAnno-def*)

lemma [*trans*] : $P' \subseteq P \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P'\ c\ Q,A$
by (*rule conseqPre*)

lemma [*trans*]: $Q \subseteq Q' \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ P\ c\ Q',A$
by (*rule conseqPost*) *blast+*

lemma [*trans*]:
$\Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ c\ Q,A \Longrightarrow (\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P'\ s\}\ c\ Q,A$
by (*rule conseqPre*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ P'\ s \longrightarrow P\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P\ s\}\ c\ Q,A \Longrightarrow \Gamma,\Theta\vdash_{/F}\ \{s.\ P'\ s\}\ c\ Q,A$
by (*rule conseqPre*) *auto*

lemma [*trans*]:
$\Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q\ s\},A \Longrightarrow (\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*trans*]:
$(\bigwedge s.\ Q\ s \longrightarrow Q'\ s) \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q\ s\},A \Longrightarrow \Gamma,\Theta\vdash_{/F}P\ c\ \{s.\ Q'\ s\},A$
by (*rule conseqPost*) *auto*

lemma [*intro?*]: $\Gamma,\Theta\vdash_{/F}\ P\ Skip\ P,A$
by (*rule Skip*) *auto*

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

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

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

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

$\Gamma,\Theta\vdash_{/F}$ *P* (*whileAnno b* {*s. undefined*} *V c*) (*P* ∩ −*b*),*A*
by (*unfold whileAnno-def*)
(*rule HoarePartialDef.While* [*THEN conseqPrePost*],*auto*)

lemma *WhileInvConj* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}$ {*s. P s* ∧ *b s*} *c* {*s. P s*},*A*
⟹ $\Gamma,\Theta\vdash_{/F}$ {*s. P s*} (*whileAnno* {*s. b s*} {*s. P s*} *V c*) {*s. P s* ∧ ¬ *b s*},*A*
by (*simp add*: *While Collect-conj-eq Collect-neg-eq*)

lemma *WhileConj* [*intro?*]:
$\Gamma,\Theta\vdash_{/F}$ {*s. P s* ∧ *b s*} *c* {*s. P s*},*A*
⟹
$\Gamma,\Theta\vdash_{/F}$ {*s. P s*} (*whileAnno* {*s. b s*} {*s. undefined*} *V c*) {*s. P s* ∧ ¬ *b s*},*A*
by (*unfold whileAnno-def*)
(*simp add*: *HoarePartialDef.While* [*THEN conseqPrePost*]
Collect-conj-eq Collect-neg-eq)

end

10 Terminating Programs

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

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

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

| *Basic*: Γ⊢*Basic f* ↓(*Normal s*)

| *Spec*: Γ⊢*Spec r* ↓(*Normal s*)

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

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

| *Fault* [*intro*,*simp*]: Γ⊢*c*↓*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* $\exists t.\ (s,t) \in r$) (*auto intro*: *exec.intros*)

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

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

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

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

done
qed
}
ultimately show P **by** (*iprover intro*: *e*)
qed

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

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

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

lemma *terminates-dynCall*:
$[\![\Gamma\vdash call\ init\ (p\ s)\ return\ c \downarrow Normal\ s]\!]$
$\Longrightarrow \Gamma\vdash dynCall\ init\ p\ return\ c \downarrow Normal\ s$

apply (*unfold dynCall-def*)
apply (*auto intro*: *terminates.intros terminates-call*)
done

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

10.2 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 $\forall s'.\ \Gamma\vdash\langle \textit{sequence Seq xs}, \textit{Normal } s'' \rangle \Rightarrow s' \longrightarrow \Gamma\vdash \textit{sequence Seq ys} \downarrow$
s'
by (*auto intro*: *exec.intros*)
from *Cons.hyps* [*OF termi-xs* [*simplified Normal*] *this*]
have $\Gamma\vdash \textit{sequence Seq } (xs\ @\ ys) \downarrow \textit{Normal } s''$.
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*:
$\bigwedge s.\ \Gamma\vdash \textit{sequence Seq } (xs\ @\ ys) \downarrow \textit{Normal } s$
$\Longrightarrow \Gamma\vdash \textit{sequence Seq xs} \downarrow \textit{Normal } s\ \wedge$
$(\forall s'.\ \Gamma\vdash\langle \textit{sequence Seq xs}, \textit{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma\vdash \textit{sequence Seq ys} \downarrow s')$
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*: $\Gamma\vdash \textit{sequence Seq } ((x\ \#\ xs)\ @\ ys) \downarrow \textit{Normal } s$ **by** *fact*
show *?case*
proof (*cases xs*)
case *Nil*
with *termi-x-xs-ys* **show** *?thesis*
by (*cases ys*)
(*auto elim*: *terminates-Normal-elim-cases exec-Normal-elim-cases*
intro: *terminates-Skip'*)
next
case *Cons*
with *termi-x-xs-ys*
obtain *termi-x*: $\Gamma\vdash x \downarrow \textit{Normal } s$ **and**
termi-xs-ys: $\forall s'.\ \Gamma\vdash\langle x, \textit{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma\vdash \textit{sequence Seq } (xs@ys) \downarrow s'$
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 $\bigwedge s.\ \Gamma\vdash$*sequence Seq* (*flatten c*)$\downarrow s \Longrightarrow \Gamma\vdash c\downarrow s$
proof (*induct c*)
  case (*Seq c1 c2*)
  have $\Gamma\vdash$*sequence Seq* (*flatten* (*Seq c1 c2*)) $\downarrow$ *s* **by** *fact*
  hence *termi-app*: $\Gamma\vdash$*sequence Seq* (*flatten c1* @ *flatten c2*) $\downarrow$ *s* **by** *simp*
  show *?case*
  proof (*cases s*)
    case (*Normal s'*)
    have $\Gamma\vdash$*Seq c1 c2* $\downarrow$ *Normal s'*
    proof (*rule terminates.Seq* [*rule-format*])
      from *termi-app* [*simplified Normal*]
      have $\Gamma\vdash$*sequence Seq* (*flatten c1*) $\downarrow$ *Normal s'*
        by (*cases rule*: *terminates-sequence-appE*)
      with *Seq.hyps*
      show $\Gamma\vdash$*c1* $\downarrow$ *Normal s'*
        by *simp*
    next
      fix *s''*
      assume $\Gamma\vdash\langle$*c1*,*Normal s'* $\rangle \Rightarrow$ *s''*
      from *termi-app* [*simplified Normal*] *exec-to-exec-sequence-flatten* [*OF this*]
      have $\Gamma\vdash$*sequence Seq* (*flatten c2*) $\downarrow$ *s''*
        by (*cases rule*: *terminates-sequence-appE*) *auto*
      with *Seq.hyps*
      show $\Gamma\vdash$*c2* $\downarrow$ *s''*
        by *simp*
    qed
    with *Normal* **show** *?thesis*
      by *simp*
  qed (*auto intro*: *terminates.intros*)
qed (*auto intro*: *terminates.intros*)

lemma *terminates-normalize-to-terminates*:
  shows $\bigwedge s.\ \Gamma\vdash$*normalize c*$\downarrow s \Longrightarrow \Gamma\vdash c\downarrow s$
proof (*induct c*)
  case *Skip* **thus** *?case* **by** (*auto intro*: *terminates-Skip'*)
next
  case *Basic* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next
  case *Spec* **thus** *?case* **by** (*cases s*) (*auto intro*: *terminates.intros*)
next

case (*Seq c1 c2*)
have 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′* ∈ *b* **and**
termi-norm-c: Γ⊢*normalize c* ↓ *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)

10.3 Lemmas about *strip-guards*

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

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

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

case (*Abrupt* t')
thus *?thesis* **by** *simp*
next
case (*Normal* t')
with *exec-c*
have $\Gamma\vdash\langle$*strip-guards F c*,*Normal s* $\rangle \Rightarrow$ *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 $\Gamma\vdash$ *While b c* $\downarrow$ *s*
by *auto*
next
case *Call* **thus** *?case* **by** *simp*
next
case *DynCom* **thus** *?case*
by (*cases s*) (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros rangeI*)
next
case *Guard*
thus *?case*
by (*cases s*) (*auto elim*: *terminates-Normal-elim-cases intro*: *terminates.intros split*: *if-split-asm*)
next
case *Throw* **thus** *?case* **by** *simp*
next
case (*Catch c1 c2*)
hence $\Gamma\vdash$*Catch* (*strip-guards F c1*) (*strip-guards F c2*) $\downarrow$ *s* **by** *simp*
thus $\Gamma\vdash$*Catch c1 c2* $\downarrow$ *s*
proof (*cases*)
fix *f* **assume** *s*=*Fault f* **thus** *?thesis* **by** *simp*
next
assume *s*=*Stuck* **thus** *?thesis* **by** *simp*
next
fix s' **assume** *s*=*Abrupt* s' **thus** *?thesis* **by** *simp*
next
fix s'
assume *s*: *s*=*Normal* s'
assume $\Gamma\vdash$*strip-guards F c1* $\downarrow$ *Normal* s'

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

10.4 Lemmas about $c_1 \cap_g c_2$

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

case (*Spec r*) **thus** *?case* **by** (*fastforce simp add*: *inter-guards-Spec*)
next
case (*Seq a1 a2*)
have (*Seq a1 a2* $\cap_g$ *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 *termi-c1*: $\Gamma\vdash$*Seq a1 a2* $\downarrow$ *s* **by** *fact*
have $\Gamma\vdash$*Seq d1 d2* $\downarrow$ *s*
proof (*cases s*)
case *Fault* **thus** *?thesis* **by** *simp*
next
case *Stuck* **thus** *?thesis* **by** *simp*
next
case *Abrupt* **thus** *?thesis* **by** *simp*
next
case (*Normal s'*)
note *Normal-s* = *this*
with *d1 termi-c1*
have $\Gamma\vdash$*d1* $\downarrow$ *Normal s'*
by (*auto elim*: *terminates-Normal-elim-cases intro*: *Seq.hyps*)
moreover
{
fix *t*
assume *exec-d1*: $\Gamma\vdash\langle$*d1*,*Normal s'* $\rangle \Rightarrow$ *t*
have $\Gamma\vdash$*d2* $\downarrow$ *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 $\Gamma\vdash\langle$*a1*,*Normal s'* $\rangle \Rightarrow$ *Normal 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*: *Seq.hyps*)
with *Normal* **show** *?thesis* **by** *simp*
qed
}
ultimately have $\Gamma\vdash$*Seq d1 d2* $\downarrow$ *Normal s'*
by (*fastforce intro*: *terminates.intros*)
with *Normal* **show** *?thesis* **by** *simp*

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

```
          with WhileTrue have Γ⊢ While b bdy ↓ Normal t'
            by simp
          with Normal show ?thesis by simp
        qed
      }
      ultimately show ?thesis
        using s-in-b
        by (blast intro: terminates.WhileTrue)
    qed
  next
    case WhileFalse thus ?case
      by (blast intro: terminates.WhileFalse)
  qed (simp-all)
}
ultimately
show ?case using c by simp
next
  case Call thus ?case by (simp add: inter-guards-Call)
next
  case (DynCom f1)
  have (DynCom f1 ∩g c2) = Some c by fact
  then obtain f2 f where
    c2: c2=DynCom f2 and
    f-defined: ∀ s. ((f1 s) ∩g (f2 s)) ≠ None and
    c: c=DynCom (λs. the ((f1 s) ∩g (f2 s)))
    by (auto simp add: inter-guards-DynCom)
  have termi: Γ⊢DynCom f1 ↓ s by fact
  show ?case
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    from f-defined obtain f where f: ((f1 s') ∩g (f2 s')) = Some f
      by auto
    from Normal termi
    have Γ⊢f1 s'↓ (Normal s')
      by (auto elim: terminates-Normal-elim-cases)
    from DynCom.hyps f this
    have Γ⊢f↓ (Normal s')
      by blast
    with c f Normal
    show ?thesis
      by (auto intro: terminates.intros)
  qed
next
```

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

have $\Gamma\vdash$*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

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

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

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

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

apply (*auto dest*: *exec-to-exec-mark-guards*)
done
ultimately show *?case*
using *s-in-b*
by (*auto intro*: *terminates.intros*)
next
case *WhileFalse* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
qed *auto*
}
with *While* **show** *?case* **by** *simp*
next
case *Call* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case*
by (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case (*Guard f g c*)
thus *?case* **by** (*fastforce intro*: *terminates.intros elim*: *terminates-Normal-elim-cases*)
next
case *Throw* **thus** *?case*
by (*fastforce intro*: *terminates.intros*)
next
case (*Catch c1 c2*)
have $\Gamma\vdash$*mark-guards f* (*Catch c1 c2*) $\downarrow$ *Normal s* **by** *fact*
then obtain
termi-merge-c1: $\Gamma\vdash$*mark-guards f c1* $\downarrow$ *Normal s* **and**
termi-merge-c2: $\forall s'$. $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow$ *Abrupt* $s' \longrightarrow$
$\Gamma\vdash$*mark-guards f 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-mark-guards* [*OF exec-c1*]
have $\Gamma\vdash\langle$*mark-guards f c1*,*Normal s* $\rangle \Rightarrow$ *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*)

10.6 Lemmas about *merge-guards*

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

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

10.7 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'*: $\Gamma\vdash c2'\downarrow t$
          by *auto*

```
        show ?thesis
        proof (cases t)
          case Fault thus ?thesis by auto
        next
          case Abrupt thus ?thesis by auto
        next
          case Stuck thus ?thesis by auto
        next
          case (Normal u)
          with noFault exec-c1' t'
          have 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': Γ⊢c2' ↓ Normal s
      by (auto elim: terminates-Normal-elim-cases)
    from False noFault have Γ⊢⟨c2',Normal s ⟩ ⇒∉Fault ‘ UNIV
```

by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
from *termi-c2′ c2-c2′ this*
have Γ⊢*c2* ↓ *Normal s*
by (*rule Cond.hyps*)
with *False c* **show** *?thesis*
by (*auto intro*: *terminates.intros*)
qed
next
case (*While b c′*)
have *noFault*: Γ⊢⟨*While b c′*,*Normal s* ⟩ ⇒∉*Fault* ‘ *UNIV* **by** *fact*
have *termi*: Γ⊢*While b c′* ↓ *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*: Γ⊢*d* ↓ *u*
assume *d*: *d* = *While b c′*
assume *noFault*: Γ⊢⟨*While b c′*,*u* ⟩ ⇒∉*Fault* ‘ *UNIV*
have Γ⊢*While b c′′* ↓ *u*
using *termi d noFault*
proof (*induct*)
case (*WhileTrue u b′ c′′′*)
have *u-in-b*: *u* ∈ *b* **using** *WhileTrue* **by** *simp*
have *termi-c′*: Γ⊢*c′* ↓ *Normal u* **using** *WhileTrue* **by** *simp*
have *noFault*: Γ⊢⟨*While b c′*,*Normal u* ⟩ ⇒∉*Fault* ‘ *UNIV* **using** *WhileTrue*
by *simp*
hence *noFault-c′*: Γ⊢⟨*c′*,*Normal u* ⟩ ⇒∉*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 Γ⊢*c′′* ↓ *Normal u*.
moreover
from *WhileTrue*
have *hyp-w*: ∀ *s′*. Γ⊢⟨*c′*,*Normal u* ⟩ ⇒ *s′* ⟶ Γ⊢⟨*While b c′*,*s′* ⟩ ⇒∉*Fault* ‘
UNIV
⟶ Γ⊢*While b c′′* ↓ *s′*
by *simp*
{
fix *v*
assume *exec-c′′*: Γ⊢⟨*c′′*,*Normal u* ⟩ ⇒ *v*
have Γ⊢*While b c′′* ↓ *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 Γ⊢C s ↓ Normal s
    by fast
  with c show ?case
```

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

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

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

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

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

10.8 Lemmas about *strip-guards*

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

then have *strip F* $\Gamma\vdash c \downarrow$ *Normal s* **by** (*simp add*: *Guard.hyps*)
with *s-in-g* **show** *?case*
  by (*auto intro*: *terminates.intros simp del*: *strip-simp*)
next
  case *GuardFault* **thus** *?case*
    by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
  case *Fault* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case (*Seq c1 s c2*)
  have *noFault-Seq*: $\Gamma\vdash\langle$*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*)
  then have *strip F* $\Gamma\vdash$*c1* $\downarrow$ *Normal s* **by** (*simp add*: *Seq.hyps*)
  moreover
  {
    fix s'
    assume *exec-strip-c1*: *strip F* $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow s'$
    have *strip F* $\Gamma\vdash$*c2* $\downarrow s'$
    proof (*cases isFault* s')
      case *True*
      thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
    next
      case *False*
      with *exec-strip-to-exec* [*OF exec-strip-c1*] *noFault-c1*
      have $*$: $\Gamma\vdash\langle$*c1*,*Normal s* $\rangle \Rightarrow s'$
        by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
      with *noFault-Seq* **have** $\Gamma\vdash\langle$*c2*,s' $\rangle \Rightarrow\notin$*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*: $s \in b$ **by** *fact*
  have *noFault-while*: $\Gamma\vdash\langle$ *While b c*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *F* **by** *fact*
  with *s-in-b* **have** *noFault-c*: $\Gamma\vdash\langle$*c*,*Normal s* $\rangle \Rightarrow\notin$*Fault* ' *F*
    by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
  then have *strip F* $\Gamma\vdash c \downarrow$ *Normal s* **by** (*simp add*: *WhileTrue.hyps*)
  moreover

{
fix s'
assume *exec-strip-c*: *strip F* $\Gamma\vdash\langle c, Normal\ s\ \rangle \Rightarrow s'$
have *strip F* $\Gamma\vdash While\ b\ c \downarrow s'$
proof (*cases isFault* s')
case *True*
thus *?thesis* **by** (*auto elim*: *isFaultE intro*: *terminates.intros*)
next
case *False*
with *exec-strip-to-exec* [*OF exec-strip-c*] *noFault-c*
have $*$: $\Gamma\vdash\langle c, Normal\ s\ \rangle \Rightarrow s'$
by (*auto simp add*: *final-notin-def elim*!: *isFaultE*)
with *s-in-b noFault-while* **have** $\Gamma\vdash\langle While\ b\ c, s'\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
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*: $\Gamma\ p = Some\ bdy$ **by** *fact*
have $\Gamma\vdash\langle Call\ p, Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **by** *fact*
with *bdy* **have** *bdy-noFault*: $\Gamma\vdash\langle bdy, Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto intro*: *exec.intros simp add*: *final-notin-def*)
then have *strip-bdy-noFault*: *strip F* $\Gamma\vdash\langle bdy, Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$
by (*auto simp add*: *final-notin-def dest*!: *exec-strip-to-exec elim*!: *isFaultE*)

from *bdy-noFault* **have** *strip F* $\Gamma\vdash bdy \downarrow Normal\ s$ **by** (*simp add*: *Call.hyps*)
from *terminates-noFault-strip-guards* [*OF this strip-bdy-noFault*]
have *strip F* $\Gamma\vdash strip\text{-}guards\ F\ bdy \downarrow Normal\ s$.
with *bdy* **show** *?case*
by (*fastforce intro*: *terminates.Call*)
next
case *CallUndefined* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Stuck* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *DynCom* **thus** *?case*
by (*auto intro*: *terminates.intros exec.intros simp add*: *final-notin-def*)
next
case *Throw* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case *Abrupt* **thus** *?case* **by** (*auto intro*: *terminates.intros*)
next
case (*Catch c1 s c2*)

have *noFault-Catch*: $\Gamma\vdash\langle Catch\ c1\ c2,Normal\ s\ \rangle \Rightarrow\notin Fault\ `\ F$ **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

10.9 Miscellaneous

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

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

lemma *wf-terminates-while*:
wf $\{(t,s).\ \Gamma\vdash(While\ b\ c)\downarrow Normal\ s \wedge s{\in}b \wedge$
$\Gamma\vdash\langle c,Normal\ s\ \rangle \Rightarrow Normal\ t\}$
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: Γ|_M⊢ c ↓ s
  assumes not-Stuck: Γ|_M⊢⟨c,s ⟩ ⇒∉{Stuck}
  shows Γ⊢ c ↓ s
using terminates-res not-Stuck
proof (induct)
  case Skip show ?case by (rule terminates.Skip)
next
  case Basic show ?case by (rule terminates.Basic)
next
  case Spec show ?case by (rule terminates.Spec)
next
  case Guard thus ?case
    by (auto intro: terminates.Guard dest: notStuck-GuardD)
next
  case GuardFault thus ?case by (auto intro: terminates.GuardFault)
next
  case Fault show ?case by (rule terminates.Fault)
next
  case (Seq c1 s c2)
  have not-Stuck: Γ|_M⊢⟨Seq c1 c2,Normal s ⟩ ⇒∉{Stuck} by fact
  hence c1-notStuck: Γ|_M⊢⟨c1,Normal s ⟩ ⇒∉{Stuck}
    by (rule notStuck-SeqD1)
  show Γ⊢Seq c1 c2 ↓ Normal s
  proof (rule terminates.Seq,safe)
    from c1-notStuck
    show Γ⊢c1 ↓ Normal s
      by (rule Seq.hyps)
  next
    fix s'
    assume exec: Γ⊢⟨c1,Normal s ⟩ ⇒ s'
    show Γ⊢c2 ↓ s'
    proof -
      from exec-to-exec-restrict [OF exec] obtain t' where
        exec-res: Γ|_M⊢⟨c1,Normal s ⟩ ⇒ t' and
        t'-notStuck: t' ≠ Stuck ⟶ t' = s'
        by blast
      show ?thesis
      proof (cases t'=Stuck)
        case True
        with c1-notStuck exec-res have False
          by (auto simp add: final-notin-def)
        thus ?thesis ..
      next
        case False
```

with *t′-notStuck* **have** t': $t'=s'$ **by** *simp*
with *not-Stuck exec-res*
have $\Gamma|_M\vdash\langle c2,s'\rangle \Rightarrow\notin\{Stuck\}$
by (*auto dest*: *notStuck-SeqD2*)
with *exec-res* t' *Seq.hyps*
show *?thesis*
by *auto*
qed
qed
qed
next
case *CondTrue* **thus** *?case*
by (*auto intro*: *terminates.CondTrue dest*: *notStuck-CondTrueD*)
next
case *CondFalse* **thus** *?case*
by (*auto intro*: *terminates.CondFalse dest*: *notStuck-CondFalseD*)
next
case (*WhileTrue s b c*)
have s: $s \in b$ **by** *fact*
have *not-Stuck*: $\Gamma|_M\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$ **by** *fact*
with *WhileTrue* **have** *c-notStuck*: $\Gamma|_M\vdash\langle c,Normal\ s\rangle \Rightarrow\notin\{Stuck\}$
by (*iprover intro*: *notStuck-WhileTrueD1*)
show *?case*
proof (*rule terminates.WhileTrue* [*OF s*],*safe*)
from *c-notStuck*
show $\Gamma\vdash c \downarrow Normal\ s$
by (*rule WhileTrue.hyps*)
next
fix s'
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow s'$
show $\Gamma\vdash While\ b\ c \downarrow s'$
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′-notStuck: $t' \neq Stuck \longrightarrow t' = s'$
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 $\Gamma|_M\vdash\langle c2,Normal\ s'\rangle \Rightarrow\notin\{Stuck\}$
    by (*auto dest*: *notStuck-CatchD2*)
  with *exec-res t′ Catch.hyps*
  show *?thesis*
    by *auto*
qed
qed
qed
qed

end

11 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* ⇒ ($'s$,$'p$,$'f$)*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

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

type-synonym ($'s$,$'p$,$'f$) *config* = ($'s$,$'p$,$'f$)*com* × ($'s$,$'f$) *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: Γ⊢(*Basic f*,*Normal s*) → (*Skip*,*Normal* (*f s*))

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

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

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

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

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

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

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

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

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

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

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

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

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

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*]:
$\Gamma\vdash(Skip,s) \to u$
$\Gamma\vdash(Guard\ f\ g\ c,s) \to u$
$\Gamma\vdash(Basic\ f,s) \to u$
$\Gamma\vdash(Spec\ r,s) \to u$
$\Gamma\vdash(Seq\ c1\ c2,s) \to u$
$\Gamma\vdash(Cond\ b\ c1\ c2,s) \to u$
$\Gamma\vdash(While\ b\ c,s) \to u$
$\Gamma\vdash(Call\ p,s) \to u$
$\Gamma\vdash(DynCom\ c,s) \to u$
$\Gamma\vdash(Throw,s) \to u$
$\Gamma\vdash(Catch\ c1\ c2,s) \to u$

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

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**
$final\ cfg = (fst\ cfg{=}Skip \lor (fst\ cfg{=}Throw \land (\exists s.\ snd\ cfg{=}Normal\ s)))$

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

11.2 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*: $\Gamma\vdash(c,s) \to (c',s')$
  shows *final* $(c,s) \Longrightarrow P$
using *step*
by *induct* (*auto simp add*: *final-def*)

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

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

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

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

lemma *SeqSteps*:
  assumes *steps*: $\Gamma\vdash cfg_1 \to^* cfg_2$
  shows $\bigwedge\ c_1\ s\ c_1'\ s'.\ [\![cfg_1 = (c_1,s); cfg_2=(c_1',s')]\!]$
    $\Longrightarrow \Gamma\vdash(Seq\ c_1\ c_2,s) \to^* (Seq\ c_1'\ c_2,\ s')$
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*: $\Gamma\vdash cfg_1 \to cfg''$ **by** *fact*
  have *steps*: $\Gamma\vdash cfg'' \to^* cfg_2$ **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

11.3 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: $\Gamma\vdash \langle c_1,\mathit{Normal}\ s\rangle \Rightarrow s'$ **by** *fact*
have *exec*-c_2: $\Gamma\vdash \langle c_2,s'\rangle \Rightarrow t$ **by** *fact*
show *?case*
proof (*cases* $\exists x.\ s'=\mathit{Abrupt}\ x$)
case *False*
from *False Seq.hyps* (*2*)
have $\Gamma\vdash (c_1,\ \mathit{Normal}\ s) \rightarrow^* (\mathit{Skip},\ s')$
by (*cases* s') *auto*
hence *seq*-c_1: $\Gamma\vdash (\mathit{Seq}\ c_1\ c_2,\ \mathit{Normal}\ s) \rightarrow^* (\mathit{Seq}\ \mathit{Skip}\ c_2,\ s')$
by (*rule SeqSteps*) *auto*
from *Seq.hyps* (*4*) **obtain** c' t' **where**

```
    steps-c2: Γ⊢ (c2, s') →* (c', t') and
    t: (case t of
          Abrupt x ⇒ if s' = t then c' = Skip ∧ t' = t
                     else c' = Throw ∧ t' = Normal x
        | - ⇒ c' = Skip ∧ t' = t)
    by auto
   note seq-c1
   also have Γ⊢ (Seq Skip c2, s') → (c2, s') by (rule step.SeqSkip)
   also note steps-c2
   finally have Γ⊢ (Seq c1 c2, Normal s) →* (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 Γ⊢ (c1, Normal s) →* (Throw, Normal x)
     by auto
   hence seq-c1: Γ⊢ (Seq c1 c2, Normal s) →* (Seq Throw c2, Normal x)
     by (rule SeqSteps) auto
   also have Γ⊢ (Seq Throw c2, Normal x) → (Throw, Normal x)
     by (rule SeqThrow)
   finally have Γ⊢ (Seq c1 c2, Normal s) →* (Throw, Normal x).
   moreover
   from exec-c2 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: Γ⊢ ⟨c,Normal s⟩ ⇒ s' by fact
 have exec-w: Γ⊢ ⟨While b c,s'⟩ ⇒ t by fact
 have b: s ∈ b by fact
 hence step: Γ⊢ (While b c,Normal s) → (Seq c (While b c),Normal s)
   by (rule step.WhileTrue)
 show ?case
 proof (cases ∃x. s'=Abrupt x)
   case False
   from False WhileTrue.hyps (3)
   have Γ⊢ (c, Normal s) →* (Skip, s')
     by (cases s') auto
   hence seq-c: Γ⊢ (Seq c (While b c), Normal s) →* (Seq Skip (While b c), s')
     by (rule SeqSteps) auto
```

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

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

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

corollary *exec-impl-steps-Normal-Abrupt*:
assumes *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow$ *Abrupt t*
shows $\Gamma\vdash(c,$*Normal s*$) \rightarrow^*$ (*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-c$_1$*'*: $\Gamma\vdash \langle c_1',Normal\ x'\rangle \Rightarrow s''$ **and**
exec-c$_2$: $\Gamma\vdash \langle c_2,s''\rangle \Rightarrow t$
by *cases*
from *Seq.hyps* (*2*) *Normal exec-c*$_1$*'* *s-Normal*
have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow s''$
by *simp*
from *exec.Seq* [*OF this exec-c*$_2$] *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-c$_1$: *redex* c_1 = *Guard f g c* **and**
fail: $x \notin g$
by *auto*
hence $\Gamma\vdash \langle redex\ c_1,Normal\ x\rangle \Rightarrow Fault\ f$
by (*auto intro*: *exec.intros*)
from *exec-redex-Fault* [*OF this*]
have $\Gamma\vdash \langle c_1,Normal\ x\rangle \Rightarrow Fault\ f$.
moreover from *Fault exec'* **have** $t{=}Fault\ f$
by (*auto intro*: *Fault-end*)
ultimately
show *?thesis*
using *s-Normal*
by (*auto intro*: *exec.intros*)
next
case *Stuck*
from *step-Stuck-end* [*OF step this*] *s-Normal*
have $(\exists r.\ redex\ c_1 = Spec\ r \wedge (\forall t.\ (x,\ t) \notin r)) \vee$
$(\exists p.\ redex\ c_1 = Call\ p \wedge \Gamma\ p = None)$
by *auto*
moreover
{
fix *r*
assume *redex* c_1 = *Spec r* **and** $(\forall t.\ (x,\ t) \notin r)$
hence $\Gamma\vdash \langle redex\ c_1,Normal\ x\rangle \Rightarrow Stuck$
by (*auto intro*: *exec.intros*)
from *exec-redex-Stuck* [*OF this*]

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

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

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

```
    with exec′
    have t=Fault f
      by (auto intro: Fault-end)
    with Fault
    show ?thesis
      by (auto intro: exec.intros)
  next
    case Stuck
    from step-Stuck [OF step this]
    have s′=Stuck.
    with exec′
    have t=Stuck
      by (auto intro: Stuck-end)
    with Stuck
    show ?thesis
      by (auto intro: exec.intros)
  qed
next
  case CatchThrow thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CatchSkip thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case FaultProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case StuckProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case AbruptProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
qed

theorem steps-Skip-impl-exec:
  assumes steps: Γ⊢(c,s) →* (Skip,t)
  shows Γ⊢⟨c,s⟩ ⇒ 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 Γ⊢ (c, s) → (c′, s′) and Γ⊢ ⟨c′,s′⟩ ⇒ 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

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

definition *inf*:: $('s,'p,'f)\ body \Rightarrow ('s,'p,'f)\ config \Rightarrow bool$
$(\text{-}\vdash\ \text{-} \rightarrow \ldots'(\infty')\ [60,80]\ 100)$ **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*)

11.5 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 c2 s)
  thus ?case
    apply (cases s)
    apply (fastforce intro: terminates.intros exec.intros
            elim: terminates-Normal-elim-cases )+
    done
next
  case (SeqThrow c2 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*:
$\forall\, i{<}k.\ \neg$ *final* (*head* (*f i*))
apply *clarify*
apply (*subgoal-tac* $i < Suc\ k$)
apply *blast*
apply *simp*
done

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

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

```
using not-fin
proof (induct k)
  case 0
  show ?case by simp
next
  case (Suc k)
  have not-fin-Suc:
    ∀ 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: ¬ Γ⊢(Throw,s) → ...(∞)
proof
  assume Γ⊢ (Throw, s) → ...(∞)
  then obtain f where
    step [rule-format]: ∀ i::nat. Γ⊢f i → f (i+1) and
    f-0: f 0 = (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: Γ⊢(Seq c1 c2,s) → ...(∞)
  shows Γ⊢(c1,s) → ...(∞) ∨
        (∃ s'. Γ⊢(c1,s) →* (Skip,s') ∧ Γ⊢(c2,s') → ...(∞))
proof −
  from inf-comp obtain f where
    step: ∀ i::nat. Γ⊢f i → f (i+1) and
    f-0: f 0 = (Seq c1 c2, s)
    by (auto simp add: inf-def)
  from f-0 have head-f-0: head (f 0) = (c1,s)
    by (simp add: head-def head-com-def)
  show ?thesis
  proof (cases ∃ i. final (head (f i)))
    case True
    define k where k = (LEAST i. final (head (f i)))
    have less-k: ∀ i<k. ¬ final (head (f i))
      apply (intro allI impI)
      apply (unfold k-def)
      apply (drule not-less-Least)
      apply auto
      done
    from infinite-computation-extract-head-Seq [OF step f-0 this]
    obtain step-head: ∀ i<k. Γ⊢ head (f i) → head (f (i + 1)) and
           conf: ∀ i<k. (∃ c' s'. f (i + 1) = (Seq c' c2, s'))
      by blast
    from True
    have final-f-k: final (head (f k))
      apply −
      apply (erule exE)
      apply (drule LeastI)
      apply (simp add: k-def)
      done
    moreover
    from f-0 conf [rule-format, of k − 1]
    obtain c' s' where f-k: f k = (Seq c' c2,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-def)
      from step
      have ∀ i. Γ⊢g i → g (i + 1)
        by (simp add: g-def)
      with g-0 have Γ⊢(Throw,s') → ...(∞)
        by (auto simp add: inf-def)
      with no-inf-Throw
      have ?thesis
        by auto
    }
    ultimately
    show ?thesis
      by (auto simp add: final-def head-def head-com-def)
  next
    case False
    then have not-fin: ∀ i. ¬ final (head (f i))
      by blast
    have ∀ i. Γ⊢head (f i) → head (f (i + 1))
    proof
      fix k
      from not-fin
      have ∀ i<(Suc k). ¬ final (head (f i))
        by simp

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

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

define k **where** $k = (LEAST\ i.\ final\ (head\ (f\ i)))$
have *less-k*: $\forall i<k.\ \neg\ final\ (head\ (f\ i))$
  apply (*intro allI impI*)
  apply (*unfold k-def*)
  apply (*drule not-less-Least*)
  apply *auto*
  done
from *infinite-computation-extract-head-Catch* [*OF step f-0 this*]
obtain *step-head*: $\forall i<k.\ \Gamma\vdash\ head\ (f\ i) \to head\ (f\ (i + 1))$ **and**
    conf: $\forall i<k.\ (\exists c'\ s'.\ f\ (i + 1) = (Catch\ c'\ c_2,\ s'))$
  by *blast*
from *True*
have *final-f-k*: $final\ (head\ (f\ k))$
  apply $-$
  apply (*erule exE*)
  apply (*drule LeastI*)
  apply (*simp add*: *k-def*)
  done
moreover
from *f-0 conf* [*rule-format*, *of* $k - 1$]
obtain $c'\ s'$ **where** *f-k*: $f\ k = (Catch\ c'\ c_2,s')$
  by (*cases k*) *auto*
moreover
from *step-head* **have** *steps-head*: $\Gamma\vdash head\ (f\ 0) \to^*\ head\ (f\ k)$
proof (*induct k*)
  case *0* **thus** *?case* **by** *simp*
next
  case (*Suc m*)
  have *step*: $\forall i<Suc\ m.\ \Gamma\vdash\ head\ (f\ i) \to head\ (f\ (i + 1))$ **by** *fact*
  hence $\forall i<m.\ \Gamma\vdash\ head\ (f\ i) \to head\ (f\ (i + 1))$
    by *auto*
  hence $\Gamma\vdash\ head\ (f\ 0) \to^*\ head\ (f\ m)$
    by (*rule Suc.hyps*)
  also from *step* [*rule-format*, *of m*]
  have $\Gamma\vdash\ head\ (f\ m) \to head\ (f\ (m + 1))$ **by** *simp*
  finally show *?case* **by** *simp*
qed
{
  assume *f-k*: $f\ k = (Catch\ Skip\ c_2,\ s')$
  with *steps-head*
  have $\Gamma\vdash(c_1,s) \to^*\ (Skip,s')$
    using *head-f-0*
    by (*simp add*: *head-def head-com-def*)
  moreover
  from *step* [*rule-format*, *of k*] *f-k*
  obtain $\Gamma\vdash(Catch\ Skip\ c_2,s') \to (Skip,s')$ **and**
   f-Suc-k: $f\ (k + 1) = (Skip,s')$
    by (*fastforce elim*: *step.cases intro*: *step.intros*)
  from *step* [*rule-format*, *of* $k+1$, *simplified f-Suc-k*]

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

from *infinite-computation-extract-head-Catch* [*OF step f-0 this*]
show $\Gamma\vdash\ head\ (f\ k) \rightarrow head\ (f\ (k + 1))$ **by** *simp*
qed
with *head-f-0* **have** $\Gamma\vdash(c_1,s) \rightarrow \ldots(\infty)$
by (*auto simp add*: *inf-def*)

thus *?thesis*
by *simp*
qed
qed

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

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

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

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

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

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

case (*Call p*)
show *?case*
proof (*rule not-infI*)
fix *f*
assume *f-step*: $\bigwedge i.\ \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$
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 c₁ c₂)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Cond b c₁ c₂, 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: ⋀i. Γ⊢f i → 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: ⋀i. Γ⊢f i → 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: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (DynCom d, Abrupt s)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Guard m g c)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
```

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


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

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

```
    by (rule not-inf-Fault)
next
  case (Seq c₁ s c₂)
  show ?case
  proof
    assume Γ⊢ (Seq c₁ c₂, Normal s) → ...(∞)
    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 ∈ b by fact
  have hyp-c1: ¬ Γ⊢ (c1, 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 = (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 Γ⊢ (c1, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. 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 ∉ b by fact
  have hyp-c2: ¬ Γ⊢ (c2, 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 = (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 Γ⊢ (c2, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp-c2 show False by simp
  qed
next
```

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

```
    from bdy f-step [of 0] f-0
    have f 1 = (bdy,Normal s)
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have Γ⊢ (bdy, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp show False by simp
  qed
next
  case (CallUndefined p s)
  have no-bdy: Γ p = None by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (Call p, Normal s)
    from no-bdy f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Stuck c)
  show ?case
    by (rule not-inf-Stuck)
next
  case (DynCom c s)
  have hyp: ¬ Γ⊢ (c s, Normal s) → ...(∞) by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → f (Suc i)
    assume f-0: f 0 = (DynCom c, Normal s)
    from f-step [of 0] f-0
    have f (Suc 0) = (c s, Normal s)
      by (auto elim: step-elim-cases)
    with f-step have Γ⊢ (c s, Normal s) → ...(∞)
      apply (simp add: inf-def)
      apply (rule-tac x=λi. f (Suc i) in exI)
      by simp
    with hyp
    show False by simp
  qed
next
  case (Throw s) thus ?case
  proof (rule not-infI)
    fix f
    assume f-step: ⋀i. Γ⊢f i → 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 $\Gamma\vdash(redex\ c,s) \rightarrow (r',s') \Longrightarrow \Gamma\vdash(c,s) \rightarrow (subst\text{-}redex\ c\ r',s')$
by (*induct c*) (*auto intro*: *step.Seq step.Catch*)

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

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

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

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

lemma *renumber'*:
assumes f: $\forall i.\ (a,f\ i) \in r^* \wedge (f\ i,f(Suc\ i)) \in r$
assumes *a-b*: $(a,b) \in r^*$
shows $b = f\ 0 \Longrightarrow (\exists f.\ f\ 0 = a \wedge (\forall i.\ (f\ i,\ f(Suc\ i)) \in r))$
using *a-b*
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
assume $b = f\ 0$
with f **show** $\exists f.\ f\ 0 = b \wedge (\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r)$
by *blast*
next
fix *a z*
assume *a-z*: $(a,\ z) \in r$ **and** $(z,\ b) \in r^*$
assume $b = f\ 0 \Longrightarrow \exists f.\ f\ 0 = z \wedge (\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r)$
$b = f\ 0$
then obtain f **where** *f0*: $f\ 0 = z$ **and** *seq*: $\forall i.\ (f\ i,\ f\ (Suc\ i)) \in r$
by *iprover*
{
fix i **have** $((\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i)\ i,\ f\ i) \in r$
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*: $\Gamma\vdash c\downarrow s$
 shows $\neg(\exists f.\ f\ 0 = (c,s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow^{+} f(Suc\ i)))$
proof −
  have $wf(\{(y,x).\ \Gamma\vdash(c,s) \rightarrow^{*} x \wedge \Gamma\vdash x \rightarrow y\}^{+})$
  proof (*rule wf-trancl*)
    show $wf\ \{(y, x).\ \Gamma\vdash(c,s) \rightarrow^{*} x \wedge \Gamma\vdash x \rightarrow y\}$
    proof (*simp only*: *wf-iff-no-infinite-down-chain*,*clarify*,*simp*)
      fix *f*
      assume $\forall i.\ \Gamma\vdash(c,s) \rightarrow^{*} f\ i \wedge \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$
      hence $\exists f.\ f\ (0{::}nat) = (c,s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i))$
        by (*rule renumber* [*to-pred*])
      moreover from *terminates-impl-no-infinite-computation* [*OF terminates*]
      have $\neg\ (\exists f.\ f\ (0{::}nat) = (c,\ s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)))$
        by (*simp add*: *inf-def*)
      ultimately show *False*
        by *simp*
    qed
  qed
  hence $\neg\ (\exists f.\ \forall i.\ (f\ (Suc\ i),\ f\ i)$
              $\in \{(y,\ x).\ \Gamma\vdash(c,\ s) \rightarrow^{*} x \wedge \Gamma\vdash x \rightarrow y\}^{+})$
    by (*simp add*: *wf-iff-no-infinite-down-chain*)
  thus *?thesis*
  proof (*rule contrapos-nn*)
    assume $\exists f.\ f\ (0{::}nat) = (c,\ s) \wedge (\forall i.\ \Gamma\vdash f\ i \rightarrow^{+} f\ (Suc\ i))$
    then obtain *f* **where**
      f0: $f\ 0 = (c,\ s)$ **and**
      seq: $\forall i.\ \Gamma\vdash f\ i \rightarrow^{+} f\ (Suc\ i)$
      by *iprover*
    show
      $\exists f.\ \forall i.\ (f\ (Suc\ i),\ f\ i) \in \{(y,\ x).\ \Gamma\vdash(c,\ s) \rightarrow^{*} x \wedge \Gamma\vdash x \rightarrow y\}^{+}$
    proof (*rule exI* [**where** *x*=*f*],*rule allI*)
      fix *i*
      show $(f\ (Suc\ i),\ f\ i) \in \{(y,\ x).\ \Gamma\vdash(c,\ s) \rightarrow^{*} x \wedge \Gamma\vdash x \rightarrow y\}^{+}$
      proof −
        {
          fix *i* **have** $\Gamma\vdash(c,s) \rightarrow^{*} f\ i$
          proof (*induct i*)
            case *0* **show** $\Gamma\vdash(c,\ s) \rightarrow^{*} f\ 0$
              by (*simp add*: *f0*)
          next

```
        case (Suc n)
        have Γ⊢(c, s) →* f n  by fact
        with seq show Γ⊢(c, s) →* f (Suc n)
          by (blast intro: tranclp-into-rtranclp rtranclp-trans)
      qed
    }
    hence Γ⊢(c,s) →* f i
      by iprover
    with seq have
      (f (Suc i), f i) ∈ {(y, x). Γ⊢(c, s) →* x ∧ Γ⊢x →+ y}
      by clarsimp
    moreover
    have ∀ y. Γ⊢f i →+ y⟶Γ⊢(c, s) →* f i⟶Γ⊢(c, s) →* 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: ∀ i. (λ(t, q) (s, p).
              Γ⊢Call p ↓ Normal s ∧
              (∃ c. Γ⊢ (Call p, Normal s) →+ (c, Normal t) ∧ 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': ∀ i. Γ⊢Call (p i) ↓ Normal (s i) ∧
            (∃ c. Γ⊢ (Call (p i), Normal (s i)) →+ (c, Normal (s (i+1))) ∧
                 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 ∃ c. ∀ i. Γ⊢Call (p i) ↓ Normal (s i) ∧
            Γ⊢ (Call (p i), Normal (s i)) →+ (c i, Normal (s (i+1))) ∧
                 redex (c i) = Call (p (i+1))
      apply −
      apply (rule choice)
```

by *blast*
then obtain *c* **where**
termi-c: $\forall\, i.\ \Gamma\vdash Call\ (p\ i) \downarrow Normal\ (s\ i)$ **and**
steps-c: $\forall\, i.\ \Gamma\vdash (Call\ (p\ i),\ Normal\ (s\ i)) \rightarrow^{+} (c\ i,\ Normal\ (s\ (i{+}1)))$ **and**
red-c: $\forall\, i.\ redex\ (c\ i) = Call\ (p\ (i{+}1))$
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 $subst\text{-}redex\ (seq\ c\ (p\ 0)\ i)\ (Call\ (p\ i)) = (seq\ c\ (p\ 0)\ i)$
by (*simp add*: *subst-redex-redex*)
} note *subst-redex-seq* = *this*
have $\forall\, i.\ \Gamma\vdash (g\ i) \rightarrow^{+} (g\ (i{+}1))$
proof
fix *i*
from *steps-c* [*rule-format*, *of i*]
have $\Gamma\vdash (Call\ (p\ i),\ Normal\ (s\ i)) \rightarrow^{+} (c\ i,\ Normal\ (s\ (i + 1)))$**.**
from *steps-redex'* [*OF this*, *of* (*seq c* (*p 0*) *i*)]
have $\Gamma\vdash (subst\text{-}redex\ (seq\ c\ (p\ 0)\ i)\ (Call\ (p\ i)),\ Normal\ (s\ i)) \rightarrow^{+}$
$(subst\text{-}redex\ (seq\ c\ (p\ 0)\ i)\ (c\ i),\ Normal\ (s\ (i + 1)))$ **.**
hence $\Gamma\vdash (seq\ c\ (p\ 0)\ i,\ Normal\ (s\ i)) \rightarrow^{+}$
$(seq\ c\ (p\ 0)\ (i{+}1),\ Normal\ (s\ (i + 1)))$
by (*simp add*: *subst-redex-seq*)
thus $\Gamma\vdash (g\ i) \rightarrow^{+} (g\ (i{+}1))$
by (*simp add*: *g-def*)
qed
moreover
from *terminates-impl-no-infinite-trans-computation* [*OF termi-c* [*rule-format*, *of 0*]]
have $\neg\ (\exists f.\ f\ 0 = (Call\ (p\ 0),\ Normal\ (s\ 0)) \wedge (\forall\, i.\ \Gamma\vdash f\ i \rightarrow^{+} f\ (Suc\ i)))$ **.**
ultimately show *False*
by *auto*
qed
qed

lemma *no-infinite-computation-implies-wf*:
assumes *not-inf*: $\neg\ \Gamma\vdash (c,\ s) \rightarrow \ldots(\infty)$
shows $wf\ \{(c2,c1).\ \Gamma \vdash (c,s) \rightarrow^{*} c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$
proof (*simp only*: *wf-iff-no-infinite-down-chain*,*clarify*, *simp*)
fix *f*
assume $\forall\, i.\ \Gamma\vdash(c,\ s) \rightarrow^{*} f\ i \wedge \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)$

hence $\exists f.\ f\ 0 = (c,\ s) \wedge (\forall\, i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i))$
  by (*rule renumber* [*to-pred*])
moreover from *not-inf*
have $\neg\ (\exists f.\ f\ 0 = (c,\ s) \wedge (\forall\, i.\ \Gamma\vdash f\ i \rightarrow f\ (Suc\ i)))$
  by (*simp add*: *inf-def*)
ultimately show *False*
  by *simp*
qed

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

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

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

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

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

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

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

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

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

$| \text{-} \Rightarrow c_f = \textit{Skip} \wedge t = s')$
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: $\Gamma\vdash (c_1,\ \textit{Normal}\ s) \rightarrow (c'',\ s'')$ **and**
rest: $\Gamma\vdash (c'',\ s'') \rightarrow^* (c_f,\ t)$
by *blast*
from *step.Seq* [*OF first*]
have $\Gamma\vdash (\textit{Seq}\ c_1\ c_2,\ \textit{Normal}\ s) \rightarrow (\textit{Seq}\ c''\ c_2,\ s'')$.
from *hyp* [*OF this*]
have *termi-s*$''$: $\Gamma\vdash \textit{Seq}\ c''\ c_2 \downarrow s''$.
show *?thesis*
proof (*cases* s'')
case (*Normal* x)
from *termi-s*$''$ [*simplified Normal*]
have *termi-c*$_2$: $\forall\, t.\ \Gamma\vdash \langle c'', \textit{Normal}\ x\rangle \Rightarrow t \longrightarrow \Gamma\vdash c_2 \downarrow t$
by *cases*
show *?thesis*
proof (*cases* $\exists\, x'.\ s' = \textit{Abrupt}\ x'$)
case *False*
with *fin* **obtain** $c_f = \textit{Skip}$ $t = s'$
by (*cases* s') *auto*
from *steps-Skip-impl-exec* [*OF rest* [*simplified this*]] *Normal*
have $\Gamma\vdash \langle c'', \textit{Normal}\ x\rangle \Rightarrow s'$
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': $s' = \textit{Abrupt}\ x'$ **and** $c_f = \textit{Throw}$ $t = \textit{Normal}$
x'
by *auto*
from *steps-Throw-impl-exec* [*OF rest* [*simplified this*]] *Normal*
have $\Gamma\vdash \langle c'', \textit{Normal}\ x\rangle \Rightarrow \textit{Abrupt}\ x'$
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 $t = \textit{Abrupt}\ x$ **by** *simp*
with *fin* **have** $s' = \textit{Abrupt}\ x$
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 $\Gamma\vdash c_2 \downarrow s'$
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 $\Gamma\vdash c_2 \downarrow s'$
by *auto*
qed
qed
qed
qed
next
case $(Cond\ b\ c_1\ c_2)$
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Cond\ b\ c_1\ c_2,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof $(cases\ s{\in}b)$
case *True*
then have $\Gamma\vdash\ (Cond\ b\ c_1\ c_2,\ Normal\ s) \rightarrow (c_1,\ Normal\ s)$
by $(rule\ step.CondTrue)$
from *hyp* $[OF\ this]$ **have** $\Gamma\vdash c_1 \downarrow Normal\ s$**.**
with *True* **show** *?thesis*
by $(auto\ intro:\ terminates.intros)$
next
case *False*
then have $\Gamma\vdash\ (Cond\ b\ c_1\ c_2,\ Normal\ s) \rightarrow (c_2,\ Normal\ s)$
by $(rule\ step.CondFalse)$
from *hyp* $[OF\ this]$ **have** $\Gamma\vdash c_2 \downarrow Normal\ s$**.**
with *False* **show** *?thesis*
by $(auto\ intro:\ terminates.intros)$
qed
next
case $(While\ b\ c)$
have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (While\ b\ c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof $(cases\ s{\in}b)$
case *True*
then have $\Gamma\vdash\ (While\ b\ c,\ Normal\ s) \rightarrow (Seq\ c\ (While\ b\ c),\ Normal\ s)$
by $(rule\ step.WhileTrue)$
from *hyp* $[OF\ this]$ **have** $\Gamma\vdash(Seq\ c\ (While\ b\ c)) \downarrow Normal\ s$**.**
with *True* **show** *?thesis*
by $(auto\ elim:\ terminates\text{-}Normal\text{-}elim\text{-}cases\ intro:\ terminates.intros)$

  next
    case *False*
    thus *?thesis*
      by (*auto intro*: *terminates.intros*)
  qed
next
  case (*Call p*)
  have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Call\ p,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  show *?case*
  proof (*cases* Γ *p*)
    case *None*
    thus *?thesis*
      by (*auto intro*: *terminates.intros*)
  next
    case (*Some bdy*)
    then have $\Gamma\vdash\ (Call\ p,\ Normal\ s) \rightarrow (bdy,\ Normal\ s)$
      by (*rule step.Call*)
    from *hyp* [*OF this*] **have** $\Gamma\vdash bdy \downarrow Normal\ s$**.**
    with *Some* **show** *?thesis*
      by (*auto intro*: *terminates.intros*)
  qed
next
  case (*DynCom c*)
  have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (DynCom\ c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  have $\Gamma\vdash\ (DynCom\ c,\ Normal\ s) \rightarrow (c\ s,\ Normal\ s)$
    by (*rule step.DynCom*)
  from *hyp* [*OF this*] **have** $\Gamma\vdash c\ s \downarrow Normal\ s$**.**
  then show *?case*
    by (*auto intro*: *terminates.intros*)
next
  case (*Guard f g c*)
  have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash\ (Guard\ f\ g\ c,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
  show *?case*
  proof (*cases* $s \in g$)
    case *True*
    then have $\Gamma\vdash\ (Guard\ f\ g\ c,\ Normal\ s) \rightarrow (c,\ Normal\ s)$
      by (*rule step.Guard*)
    from *hyp* [*OF this*] **have** $\Gamma\vdash c\downarrow Normal\ s$**.**
    with *True* **show** *?thesis*
      by (*auto intro*: *terminates.intros*)
  next
    case *False*
    thus *?thesis*
      by (*auto intro*: *terminates.intros*)
  qed
next
  case *Throw* **show** *?case* **by** (*auto intro*: *terminates.intros*)
next
  case (*Catch* c_1 c_2)

have *hyp*: $\bigwedge c'\ s'.\ \Gamma\vdash (Catch\ c_1\ c_2,\ Normal\ s) \rightarrow (c',\ s') \Longrightarrow \Gamma\vdash c' \downarrow s'$ **by** *fact*
show *?case*
proof (*rule terminates.Catch*)
  {
    fix $c'\ s'$
    assume *step-c*$_1$: $\Gamma\vdash (c_1,\ Normal\ s) \rightarrow (c',\ s')$
    have $\Gamma\vdash c' \downarrow s'$
    proof −
      from *step-c*$_1$
      have $\Gamma\vdash (Catch\ c_1\ c_2,\ Normal\ s) \rightarrow (Catch\ c'\ c_2,\ s')$
        by (*rule step.Catch*)
      from *hyp* [*OF this*]
      have $\Gamma\vdash Catch\ c'\ c_2 \downarrow s'$.
      thus $\Gamma\vdash c' \downarrow s'$
        by *cases auto*
    qed
  }
  from *Catch.hyps* (*1*) [*OF this*]
  show $\Gamma\vdash c_1 \downarrow Normal\ s$.
next
  show $\forall s'.\ \Gamma\vdash \langle c_1, Normal\ s\rangle \Rightarrow Abrupt\ s' \longrightarrow \Gamma\vdash c_2 \downarrow Normal\ s'$
  proof (*intro allI impI*)
    fix s'
    assume *exec-c*$_1$: $\Gamma\vdash \langle c_1, Normal\ s\rangle \Rightarrow Abrupt\ s'$
    show $\Gamma\vdash c_2 \downarrow Normal\ s'$
    proof (*cases final* $(c_1, Normal\ s)$)
      case *True*
      with *exec-c*$_1$
      have *Throw*: $c_1 = Throw$
        by (*auto simp add*: *final-def elim*: *exec-Normal-elim-cases*)
      have $\Gamma\vdash (Catch\ Throw\ c_2, Normal\ s) \rightarrow (c_2, Normal\ s)$
        by (*rule step.CatchThrow*)
      from *hyp* [*simplified Throw*, *OF this*]
      have $\Gamma\vdash c_2 \downarrow Normal\ s$.
      moreover from *exec-c*$_1$ *Throw*
      have $s' = s$
        by (*auto elim*: *exec-Normal-elim-cases*)
      ultimately show *?thesis* **by** *simp*
    next
      case *False*
      from *exec-impl-steps* [*OF exec-c*$_1$]
      obtain $c_f\ t$ **where**
        steps-c$_1$: $\Gamma\vdash (c_1,\ Normal\ s) \rightarrow^* (Throw,\ Normal\ s')$
        by (*fastforce split*: *xstate.splits*)
      from *split-computation* [*OF steps-c*$_1$ *False*]
      obtain $c''\ s''$ **where**
        first: $\Gamma\vdash (c_1,\ Normal\ s) \rightarrow (c'',\ s'')$ **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* $\{(c2,\ c1).\ \Gamma\vdash(c,\ s) \rightarrow^* c1 \wedge \Gamma\vdash c1 \rightarrow c2\}$**.**
  show *?thesis*
    by (*rule wf-implies-termi-reach* [*OF wf*]) *auto*
qed

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

11.6 Generalised Redexes

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

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

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

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

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

lemma *step-redexes*:

shows $\bigwedge r\ r'.\ [\![\Gamma\vdash(r,s) \to (r',s');\ r \in \mathit{redexes}\ c]\!]$
$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \to (c',s') \land r' \in \mathit{redexes}\ c'$
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*: $r = \mathit{Seq}\ c_1\ c_2 \lor r \in \mathit{redexes}\ c_1$
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 $c_2 \lor r \in$ *redexes* c_1
by *simp*
have *step-r*: $\Gamma\vdash (r,\ s) \to (r',\ s')$ **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: $\Gamma\vdash (c_1,\ s) \to (c',\ s')$ **and**
r': $r' \in$ *redexes* c'
by *blast*
from *step.Catch* [*OF step*-c_1]
have $\Gamma\vdash$ (*Catch* c_1 c_2, s) $\to$ (*Catch* c' c_2, s').
with r'
show *?case*
by *auto*
qed
qed

lemma *steps-redexes*:
assumes *steps*: $\Gamma\vdash (r,\ s) \to^* (r',\ s')$
shows $\bigwedge c.\ r \in$ *redexes* $c \Longrightarrow \exists c'.\ \Gamma\vdash(c,s) \to^* (c',s') \land r' \in$ *redexes* c'
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl*
then
show $\exists c'.\ \Gamma\vdash (c,\ s') \to^* (c',\ s') \land r' \in$ *redexes* c'
by *auto*
next
case (*Trans r s* r'' s'')
have $\Gamma\vdash (r,\ s) \to (r'',\ s'')$ $r \in$ *redexes* c **by** *fact+*
from *step-redexes* [*OF this*]
obtain c' **where**
step: $\Gamma\vdash (c,\ s) \to (c',\ s'')$ **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 $\bigwedge c.\ r \in \mathit{redexes}\ c \Longrightarrow \exists c'.\ \Gamma\vdash(c,s) \rightarrow^+ (c',s') \wedge r' \in \mathit{redexes}\ c'$
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 $\exists c'.\ \Gamma\vdash(c,s) \rightarrow (c',s') \wedge \mathit{Seq}\ r'\ c_2 \in \mathit{redexes}\ c'$
proof −
from *step.Seq* [*OF step*]

have $\Gamma\vdash$ $(Seq\ r\ c_2,\ s) \to (Seq\ r'\ c_2,\ s')$.
from *step-redexes* [*OF this Seq*]
show *?thesis* .
qed

lemma *steps-redexes-Seq*:
assumes *steps*: $\Gamma\vdash (r,\ s) \to^* (r',\ s')$
shows $\bigwedge c.\ Seq\ r\ c_2 \in redexes\ c \Longrightarrow$
$\exists\, c'.\ \Gamma\vdash(c,s) \to^* (c',s') \wedge Seq\ r'\ c_2 \in redexes\ c'$
using *steps*
proof (*induct rule*: *converse-rtranclp-induct2* [*case-names Refl Trans*])
case *Refl*
then show *?case*
by (*auto*)

next
case (*Trans* $r\ s\ r''\ s''$)
have $\Gamma\vdash (r,\ s) \to (r'',\ s'')\ Seq\ r\ c_2 \in redexes\ c$ **by** *fact+*
from *step-redexes-Seq* [*OF this*]
obtain c' **where**
step: $\Gamma\vdash (c,\ s) \to (c',\ s'')$ **and**
r'': $Seq\ r''\ c_2 \in redexes\ c'$
by *blast*
note *step*
also
from *Trans.hyps* (*3*) [*OF* r'']
obtain c'' **where**
steps: $\Gamma\vdash (c',\ s'') \to^* (c'',\ s')$ **and**
r': $Seq\ r'\ c_2 \in redexes\ c''$
by *blast*
note *steps*
finally
show *?case*
using r'
by *blast*
qed

lemma *steps-redexes-Seq'*:
assumes *steps*: $\Gamma\vdash (r,\ s) \to^+ (r',\ s')$
shows $\bigwedge c.\ Seq\ r\ c_2 \in redexes\ c$
$\Longrightarrow \exists\, c'.\ \Gamma\vdash(c,s) \to^+ (c',s') \wedge Seq\ r'\ c_2 \in redexes\ c'$
using *steps*
proof (*induct rule*: *tranclp-induct2* [*consumes 1*, *case-names Step Trans*])
case (*Step* $r'\ s'\ c'$)
have $\Gamma\vdash (r,\ s) \to (r',\ s')\ Seq\ r\ c_2 \in redexes\ c'$ **by** *fact+*
from *step-redexes-Seq* [*OF this*]
show *?case*
by (*blast intro*: *r-into-trancl*)
next

case (*Trans* r' s' r'' s'')
from *Trans* **obtain** c' **where**
steps: $\Gamma\vdash$ $(c,\ s) \to^+ (c',\ s')$ **and**
r': *Seq* r' c_2 $\in$ *redexes* c'
by *blast*
note *steps*
moreover
have $\Gamma\vdash$ $(r',\ s') \to (r'',\ s'')$ **by** *fact*
from *step-redexes-Seq* [*OF this* r'] **obtain** c'' **where**
step: $\Gamma\vdash$ $(c',\ s') \to (c'',\ s'')$ **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*: $\Gamma\vdash(r,s) \to (r',s')$
assumes *Catch*: *Catch* r c_2 $\in$ *redexes* c
shows $\exists\, c'.\ \Gamma\vdash(c,s) \to (c',s') \wedge$ *Catch* r' c_2 $\in$ *redexes* c'
proof −
from *step.Catch* [*OF step*]
have $\Gamma\vdash$ (*Catch* r c_2, s) $\to$ (*Catch* r' c_2, s')**.**
from *step-redexes* [*OF this Catch*]
show *?thesis* **.**
qed

lemma *steps-redexes-Catch*:
assumes *steps*: $\Gamma\vdash$ $(r,\ s) \to^* (r',\ s')$
shows $\bigwedge c.$ *Catch* r c_2 $\in$ *redexes* c $\Longrightarrow$
$\exists\, c'.\ \Gamma\vdash(c,s) \to^* (c',s') \wedge$ *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 $\Gamma\vdash$ $(r,\ s) \to (r'',\ s'')$ *Catch* r c_2 $\in$ *redexes* c **by** *fact+*
from *step-redexes-Catch* [*OF this*]
obtain c' **where**
step: $\Gamma\vdash$ $(c,\ s) \to (c',\ s'')$ **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

12 Hoare Logic for Total Correctness

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

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

definition
validt :: $[('s,'p,'f)\ body,'f\ set,'s\ assn,('s,'p,'f)\ com,'s\ assn,'s\ assn] \Rightarrow bool$
(-$\models_{t'/\_}$/ - - -,- [61,60,1000, 20, 1000,1000] 60)
where
$\Gamma \models_{t/F} P\ c\ Q,A \equiv \Gamma \models_{/F} P\ c\ Q,A \land (\forall s \in Normal\ `\ P.\ \Gamma \vdash c \downarrow s)$

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

notation (*ASCII*)
validt (-|=t'/-/ - - -,- [61,60,1000, 20, 1000,1000] 60) **and**
cvalidt (-,-|=t'/- / - - -,- [61,60,60,1000, 20, 1000,1000] 60)

12.2 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*)

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

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

| *DynCom*: $\forall s \in P.\ \Gamma,\Theta\vdash_{t/F} P\ (c\ s)\ Q,A$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (\mathit{DynCom}\ c)\ Q,A$

| *Throw*: $\Gamma,\Theta\vdash_{t/F} A\ \mathit{Throw}\ Q,A$

| *Catch*: $[\![\Gamma,\Theta\vdash_{t/F} P\ c_1\ Q,R; \Gamma,\Theta\vdash_{t/F} R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ \mathit{Catch}\ c_1\ c_2\ Q,A$

| *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*: $(P,p,Q,A) \in \Theta$
$\Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (\mathit{Call}\ p)\ Q,A$

| *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* $(\Gamma\ p)$) $Q,A\ \wedge$
$(\forall x.\ \Theta \cup$ *Specs-wf p* τ
$\subseteq x \longrightarrow$
$\Gamma,x\vdash_{t/F} (\{\tau\} \cap P)$ (*the* $(\Gamma\ p)$) $Q,A)$) **by** *fact*
hence $\forall (P,p,Q,A)\in$*Specs*. $p \in$ *dom* $\Gamma\ \wedge$

$(\forall \tau.\ \Gamma,\Theta' \cup$ *Specs-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* $p\ \tau$ **in** *allE*)
apply (*insert aug*)
apply *auto*
done
with *CallRec* **show** *?case* **by** − (*rule hoaret.CallRec*)
next
case *DynCom* **thus** *?case* **by** (*blast intro*: *hoaret.DynCom*)
next
case (*Conseq* $P\ \Theta\ F\ c\ Q\ A\ \Theta'$)
from *Conseq*
have $\forall s \in P.\ (\exists P'\ Q'\ A'.\ (\Gamma,\Theta' \vdash_{t/F} P'\ c\ Q',A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
with *Conseq* **show** *?case* **by** − (*rule hoaret.Conseq*)
next
case (*ExFalso* $\Theta\ F\ P\ \ c\ Q\ A\ \Theta'$)
have $\Gamma,\Theta \models_{t/F} P\ c\ Q,A\ \neg\ \Gamma \models_{t/F} P\ c\ Q,A\ \Theta \subseteq \Theta'$ **by** *fact*+
then show *?case*
by (*fastforce intro*: *hoaret.ExFalso simp add*: *cvalidt-def*)
qed (*blast intro*: *hoaret.intros*)+

12.4 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

13 Properties of Total Correctness Hoare Logic

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

13.1 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 (*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*:
  $\forall (P,\ p,\ Q,\ A){\in}Specs.\ p \in dom\ \Gamma \land$
    $(\forall \tau.\ \Gamma,\Theta \cup Specs\text{-}wf\ p\ \tau{\models_{t/F}} (\{\tau\} \cap P)\ the\ (\Gamma\ p)\ Q,A)$ **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*:
  $\Gamma \models_{t/F} (\{s\} \cap P')$ *the* $(\Gamma\ q)$ Q',A'
  by − (*rule WF.hyps*, *auto*)
from *Pre q*
have *Pre'*: $s \in \{s\} \cap P'$
  by *auto*
from *exec-q* **show** $t \in$ *Normal* ‘ $Q' \cup$ *Abrupt* ‘ A'
proof (*cases*)
  fix *bdy*
  assume *bdy*: $\Gamma\ q =$ *Some bdy*
  assume *exec-bdy*: $\Gamma \vdash \langle bdy, Normal\ s\rangle \Rightarrow t$
  from *valid-bdy* [*simplified bdy option.sel*] *t-notin-F exec-bdy Pre'*
  have $t \in$ *Normal* ‘ $Q' \cup$ *Abrupt* ‘ A'
    by (*auto simp add*: *validt-def valid-def*)
  with *Pre q*
  show *?thesis*
    by *auto*
next
  assume $\Gamma\ q =$ *None*
  with *q valid-body* **have** *False* **by** *auto*
  thus *?thesis* **..**
qed
next
  fix *s*
  assume *Pre*: $s \in P' \cap \{s.\ ((s,q),\ \tau,p) \in r\}$
  from *Pre q* τp
  have *valid-bdy*:
    $\Gamma \models_{t/F} (\{s\} \cap P')$ (*the* $(\Gamma\ q)$) Q',A'
    by − (*rule WF.hyps*, *auto*)
  from *Pre q*
  have *Pre'*: $s \in \{s\} \cap P'$
    by *auto*
  from *valid-bdy ctxt Pre'*
  have $\Gamma \vdash$ *the* $(\Gamma\ q) \downarrow$ *Normal s*
    by (*auto simp add*: *validt-def*)
  with *valid-body q*
  show $\Gamma \vdash$ *Call* $q\downarrow$ *Normal s*
    by (*fastforce intro*: *terminates.Call*)
qed
}
hence $\forall (P,\ p,\ Q,\ A) \in$ *Specs-wf p* τ. $\Gamma \models_{t/F} P$ *Call p* Q,A
  by (*auto simp add*: *cvalidt-def Specs-wf*)
with *ctxt* **have** $\forall (P,\ p,\ Q,\ A) \in \Theta \cup$ *Specs-wf p* τ. $\Gamma \models_{t/F} P$ *Call p* Q,A
  by *auto*
with *p valid-body*
show $\Gamma \models_{t/F} (\{\tau\} \cap P)$ (*the* $(\Gamma\ p)$) 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 Θ F c Q A)
hence *adapt*:
$\forall s \in P.\ (\exists P'\ Q'\ A'.\ (\Gamma,\Theta \models_{t/F} P'\ c\ Q',A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$
by *blast*
show $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
proof (*rule cvalidtI*)
fix s t
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q,A$
assume *exec*: $\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow t$
assume P: $s \in P$
assume *t-notin-F*: $t \notin \mathit{Fault}$ ‘ F
show $t \in \mathit{Normal}$ ‘ $Q \cup \mathit{Abrupt}$ ‘ 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′*: $\mathit{Normal}\ s \in \mathit{Normal}$ ‘ $P' \longrightarrow$
$t \in \mathit{Normal}$ ‘ $Q' \cup \mathit{Abrupt}$ ‘ A'
by (*unfold cvalidt-def validt-def valid-def*) *blast*
hence $s \in P' \longrightarrow t \in \mathit{Normal}$ ‘ $Q' \cup \mathit{Abrupt}$ ‘ A'
by *blast*
with *weaken*
show *?thesis*
by *blast*
qed
next
fix s
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P\ (\mathit{Call}\ p)\ Q,A$
assume P: $s \in P$
show $\Gamma\vdash c \downarrow \mathit{Normal}\ s$
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 Θ F)
assume $(P,\ p,\ Q,\ A) \in \Theta$
then show $\Gamma,\Theta \models_{t/F} P\ (\mathit{Call}\ p)\ Q,A$

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

13.2 Completeness

lemma *MGT-valid*:
$\Gamma\models_{t/F}$ $\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$
$s\}$ c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$, $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule validtI*)
fix *s t*
assume $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
$s \in \{s.\ s = Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$
$s\}$
$t \notin Fault\ `\ F$
thus $t \in Normal\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\} \cup$
$Abrupt\ `\ \{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*cases t*)
apply (*auto simp add*: *final-notin-def*)
done
next
fix *s*
assume $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash c\downarrow Normal$
$s\}$
thus $\Gamma\vdash c\downarrow Normal\ s$
by *blast*
qed

The consequence rule where the existential Z is instantiated to s. Usefull in proof of *MGT-lemma*.

lemma *ConseqMGT*:
  assumes *modif*: $\forall Z{::}'a.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z{::}'a\ assn)\ c\ (Q'\ Z),(A'\ Z)$
  assumes *impl*: $\bigwedge s.\ s \in P \Longrightarrow s \in P'\ s \wedge (\forall t.\ t \in Q'\ s \longrightarrow t \in Q) \wedge$
  $(\forall t.\ t \in A'\ s \longrightarrow t \in A)$
  shows $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
using *impl*
by $-$ (*rule conseq* [*OF modif*],*blast*)

lemma *MGT-implies-complete*:
  assumes *MGT*: $\forall Z.\ \Gamma,\{\}\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault$
$`\ (-F)) \wedge$
  $\Gamma\vdash c\downarrow Normal\ s\}$
  c
  $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},$
  $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  assumes *valid*: $\Gamma \models_{t/F} P\ c\ Q,A$
  shows $\Gamma,\{\} \vdash_{t/F} P\ c\ Q,A$
  using *MGT*
  apply (*rule ConseqMGT*)
  apply (*insert valid*)
  apply (*auto simp add*: *validt-def valid-def intro*!: *final-notinI*)
  done

lemma *conseq-extract-state-indep-prop*:
  assumes *state-indep-prop*:$\forall s \in P.\ R$
  assumes *to-show*: $R \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
  shows $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
  apply (*rule Conseq*)
  apply (*clarify*)
  apply (*rule-tac x=P* **in** *exI*)
  apply (*rule-tac x=Q* **in** *exI*)
  apply (*rule-tac x=A* **in** *exI*)
  using *state-indep-prop to-show*
  by *blast*

lemma *MGT-lemma*:
  assumes *MGT-Calls*:
    $\forall p \in dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F}$
      $\{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
        $\Gamma\vdash(Call\ p)\downarrow Normal\ s\}$
        $(Call\ p)$
      $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
      $\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
  shows $\bigwedge Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
  $\Gamma\vdash c\downarrow Normal\ s\}$
  c
  $\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$

proof (*induct c*)
  case *Skip*
  show $\Gamma,\Theta\vdash_{t/F}$ {*s*. *s* = *Z* ∧ 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 Γ,Θ$\vdash_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*Seq c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*Seq c1 c2*↓*Normal s*} *c1*
{*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t* ∧ Γ⊢⟨*c2*,*Normal t*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*c2*↓*Normal t*},
{*t*. Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
by (*rule ConseqMGT*)
(*auto dest*: *Seq-NoFaultStuckD1* [*simplified*] *Seq-NoFaultStuckD2* [*simplified*]
elim: *terminates-Normal-elim-cases*
intro: *exec.intros*)
thus Γ,Θ$\vdash_{t/F}$ {*s*. *s*=*Z* ∧ Γ⊢⟨*Seq c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧
Γ⊢*Seq c1 c2*↓*Normal s*}
Seq c1 c2
{*t*. Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *Normal t*},
{*t*. Γ⊢⟨*Seq c1 c2*,*Normal Z*⟩ ⇒ *Abrupt t*}
proof (*rule hoaret.Seq*)
show Γ,Θ$\vdash_{t/F}$ {*t*. Γ⊢⟨*c1*,*Normal Z*⟩ ⇒ *Normal t* ∧
Γ⊢⟨*c2*,*Normal t*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧ Γ⊢*c2* ↓ *Normal t*}
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 (*rule 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 (*rule exec.intros*)
qed
qed
next
case (*Cond b c1 c2*)
have ∀ *Z*. Γ,Θ$\vdash_{t/F}$ {*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 *Cond.hyps* **by** *iprover*
hence Γ,Θ $\vdash_{t/F}$ ({*s*. *s*=*Z* ∧ Γ⊢⟨*Cond b c1 c2*,*Normal s*⟩ ⇒∉({*Stuck*} ∪ *Fault* ‘ (−*F*)) ∧

$\Gamma\vdash(\mathit{Cond}\ b\ c1\ c2)\downarrow\mathit{Normal}\ s\}\cap b)$
$c1$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondTrue*
elim: *terminates-Normal-elim-cases*)
moreover
have $\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash c2\downarrow\mathit{Normal}\ s\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
using *Cond.hyps* **by** *iprover*
hence $\Gamma,\Theta\vdash_{t/F} (\{s.\ s=Z \wedge \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F)) \wedge$
$\Gamma\vdash(\mathit{Cond}\ b\ c1\ c2)\downarrow\mathit{Normal}\ s\}\cap-b)$
$c2$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule ConseqMGT*)
(*fastforce simp add*: *final-notin-def intro*: *exec.CondFalse*
elim: *terminates-Normal-elim-cases*)
ultimately
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F)) \wedge$
$\Gamma\vdash(\mathit{Cond}\ b\ c1\ c2)\downarrow\mathit{Normal}\ s\}$
$(\mathit{Cond}\ b\ c1\ c2)$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle\mathit{Cond}\ b\ c1\ c2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
by (*rule hoaret.Cond*)
next
case (*While b c*)
let $?unroll = (\{(s,t).\ s\in b \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\})^*$
let $?P' = \lambda Z.\ \{t.\ (Z,t)\in ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle\mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ t\}$
let $?A = \lambda Z.\ \{t.\ \Gamma\vdash\langle\mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
let $?r = \{(t,s).\ \Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ s \wedge s\in b \wedge$
$\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\}$
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle\mathit{While}\ b\ c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow\mathit{Normal}\ s\}$
$(\mathit{While}\ b\ c)$
$\{t.\ \Gamma\vdash\langle\mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$

$\{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
proof (*rule ConseqMGT* [**where** $?P'=\lambda\ Z.\ ?P'\ Z$
and $?Q'=\lambda\ Z.\ ?P'\ Z \cap -\ b$])
have *wf-r*: *wf ?r* **by** (*rule wf-terminates-while*)
show $\forall\ Z.\ \Gamma,\Theta\vdash_{t/F}\ (?P'\ Z)\ (\mathit{While}\ b\ c)\ (?P'\ Z \cap -\ b),(?A\ Z)$
proof (*rule allI*, *rule hoaret.While* [*OF wf-r*])
fix *Z*
from *While*
have *hyp-c*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `$
$(-F)) \wedge$
$\Gamma\vdash c\downarrow \mathit{Normal}\ s\}$
c
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\}$,
$\{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$ **by** *iprover*
show $\forall\sigma.\ \Gamma,\Theta\vdash_{t/F}\ (\{\sigma\} \cap\ ?P'\ Z\ \cap b)\ c$
$(\{t.\ (t,\ \sigma) \in\ ?r\} \cap\ ?P'\ Z),(?A\ Z)$
proof (*rule allI*, *rule ConseqMGT* [*OF hyp-c*])
fix σ s
assume $s\in \{\sigma\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow \mathit{Normal}\ t\}$
$\cap\ b$
then obtain
s-eq-σ: $s=\sigma$ **and**
Z-s-unroll: $(Z,s) \in\ ?unroll$ **and**
noabort: $\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)$ **and**
while-term: $\Gamma\vdash(\mathit{While}\ b\ c)\downarrow \mathit{Normal}\ s$ **and**
s-in-b: $s\in b$
by *blast*
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash c\downarrow \mathit{Normal}\ t\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\} \longrightarrow$
$t \in \{t.\ (t,\sigma) \in\ ?r\} \cap$
$\{t.\ (Z,\ t) \in\ ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow\ e\in b$
$\longrightarrow \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u)) \wedge$
$\Gamma\vdash(\mathit{While}\ b\ c)\downarrow \mathit{Normal}\ t\})\ \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Abrupt}\ t\} \longrightarrow$
$t \in \{t.\ \Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\})$
(**is** $?C1 \wedge ?C2 \wedge ?C3$)

proof (*intro conjI*)
  from *Z-s-unroll noabort s-in-b while-term* **show** *?C1*
    by (*blast elim*: *terminates-Normal-elim-cases*)
next
  {
    fix *t*
    assume *s-t*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Normal\ t$
    with *s-eq-σ while-term s-in-b* **have** $(t,\sigma) \in\ ?r$
      by *blast*
    moreover
    from *Z-s-unroll s-t s-in-b*
    have $(Z,\ t) \in\ ?unroll$
      by (*blast intro*: *rtrancl-into-rtrancl*)
    moreover from *while-term s-t s-in-b*
    have $\Gamma\vdash(While\ b\ c)\downarrow Normal\ t$
      by (*blast elim*: *terminates-Normal-elim-cases*)
    moreover note *noabort*
    ultimately
    have $(t,\sigma) \in\ ?r \wedge (Z,\ t) \in\ ?unroll \wedge$
          $(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
                $\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                    $(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
                          $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
          $\Gamma\vdash(While\ b\ c)\downarrow Normal\ t$
      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*: $s \in \{s.\ s=Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
          $\Gamma\vdash While\ b\ c \downarrow Normal\ s\}$
  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 ∈ ?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 $\forall\, s \in \{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow\ Normal\ s\}.$
$p \in dom\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show *?case*
proof (*rule conseq-extract-state-indep-prop*)
assume *p-defined*: $p \in dom\ \Gamma$
with *MGT-Calls* **show**
$\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle Call\ p\ , Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\wedge$
$\Gamma\vdash Call\ \ p\downarrow Normal\ s\}$
$(Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*auto*)
qed
next
case (*DynCom c*)
have *hyp*:
$\bigwedge s'.\ \forall\, Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle c\ s', Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash c\ s'\downarrow Normal\ s\}\ c\ s'$
$\{t.\ \Gamma\vdash\langle c\ s', Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c\ s', Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp'*:
$\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle DynCom\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash DynCom\ c\downarrow Normal\ s\}$
$(c\ Z)$
$\{t.\ \Gamma\vdash\langle DynCom\ c, Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle DynCom\ c, Normal\ Z\rangle$
$\Rightarrow Abrupt\ t\}$
by (*rule ConseqMGT* [*OF hyp*])
(*fastforce simp add*: *final-notin-def intro*: *exec.intros*
elim: *terminates-Normal-elim-cases*)
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle DynCom\ c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash DynCom\ c\downarrow Normal\ s\}$
$DynCom\ c$
$\{t.\ \Gamma\vdash\langle DynCom\ c, Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle DynCom\ c, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoaret.DynCom*)
apply (*clarsimp*)
apply (*rule hyp'* [*simplified*])
done
next
case (*Guard f g c*)
have *hyp-c*: $\forall\, Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash c\downarrow Normal\ s\}$

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 $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Throw,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                 $\Gamma\vdash Throw \downarrow Normal\ s\}$
            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 $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle c_1,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                   $\Gamma\vdash c_1 \downarrow Normal\ s\}$
              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*
  hence $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                $\Gamma\vdash Catch\ c_1\ c_2 \downarrow Normal\ s\}$
            c_1
            $\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
            $\{t.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ t \wedge \Gamma\vdash c_2 \downarrow Normal\ t \wedge$
                $\Gamma\vdash\langle c_2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
    by (*rule ConseqMGT*)
       (*fastforce intro*: *exec.intros terminates.intros*
              elim: *terminates-Normal-elim-cases*
              simp add: *final-notin-def*)
  moreover
  have
    $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$ $\{s.\ s{=}Z \wedge \Gamma\vdash\langle c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                 $\Gamma\vdash c_2 \downarrow Normal\ s\}$ 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*
  hence $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ \Gamma\vdash\langle c_1,Normal\ Z\rangle \Rightarrow Abrupt\ s \wedge \Gamma\vdash c_2 \downarrow Normal\ s \wedge$
                $\Gamma\vdash\langle c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\}$
            c_2
            $\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
            $\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
      by (*rule ConseqMGT*)
         (*fastforce intro*: *exec.intros terminates.intros*
              simp add: *noFault-def'*)
  ultimately
  show $\Gamma,\Theta\vdash_{t/F}$ $\{s.\ s = Z \wedge \Gamma\vdash\langle Catch\ c_1\ c_2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
                $\Gamma\vdash Catch\ c_1\ c_2 \downarrow Normal\ s\}$

$\quad\quad$ *Catch* c_1 c_2
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Catch\ c_1\ c_2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
$\quad$ **by** (*rule hoaret.Catch*)
qed

lemma *Call-lemma'*:
assumes *Call-hyp*:
$\forall q\in dom\ \Gamma.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\quad\quad\quad\quad$ $\Gamma\vdash Call\ q\downarrow Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$\quad\quad$ $(Call\ q)$
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$\quad$ $\{s.\ s=Z \wedge \Gamma\vdash\langle c, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge \Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$\quad\quad$ $(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge c \in redexes\ c')\}$
$\quad\quad$ c
$\quad$ $\{t.\ \Gamma\vdash\langle c, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\quad$ $\{t.\ \Gamma\vdash\langle c, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*induct c*)
case *Skip*
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s = Z \wedge \Gamma\vdash\langle Skip, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\quad\quad$ $\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$\quad\quad$ $(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge Skip \in redexes\ c')\}$
$\quad\quad$ *Skip*
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Skip, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Skip, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
$\quad$ **by** (*rule hoaret.Skip* [*THEN conseqPre*]) (*blast intro*: *exec.Skip*)
next
case (*Basic f*)
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Basic\ f, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\quad\quad$ $\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$\quad\quad$ $(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge$
$\quad\quad\quad$ $Basic\ f \in redexes\ c')\}$
$\quad\quad$ *Basic f*
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Basic\ f, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\quad\quad$ $\{t.\ \Gamma\vdash\langle Basic\ f, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
$\quad$ **by** (*rule hoaret.Basic* [*THEN conseqPre*]) (*blast intro*: *exec.Basic*)
next
case (*Spec r*)
show $\Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Spec\ r, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$

$\quad\quad$ $\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$\quad\quad$ $(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge$
$\quad\quad$ $Spec\ r \in redexes\ c')\}$

Spec r
$\{t.\ \Gamma\vdash\langle Spec\ r, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Spec\ r, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule hoaret.Spec* [*THEN conseqPre*])
apply (*clarsimp*)
apply (*case-tac* $\exists t.\ (Z,t) \in r$)
apply (*auto elim*: *exec-elim-cases simp add*: *final-notin-def intro*: *exec.intros*)
done
next
case (*Seq c1 c2*)
have *hyp-c1*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c1, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge c1 \in redexes\ c')\}$
c1
$\{t.\ \Gamma\vdash\langle c1, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c1, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Seq.hyps* **by** *iprover*
have *hyp-c2*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge c2 \in redexes\ c')\}$
c2
$\{t.\ \Gamma\vdash\langle c2, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Seq.hyps* (*2*) **by** *iprover*
have *c1*: $\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Seq\ c1\ c2, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `$
$(-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge$
$Seq\ c1\ c2 \in redexes\ c')\}$
c1
$\{t.\ \Gamma\vdash\langle c1, Normal\ Z\rangle \Rightarrow Normal\ t \wedge$
$\Gamma\vdash\langle c2, Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ t) \wedge$
$c2 \in redexes\ c')\}$,
$\{t.\ \Gamma\vdash\langle Seq\ c1\ c2, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c1*],*clarify*,*safe*)
assume $\Gamma\vdash\langle Seq\ c1\ c2, Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma\vdash\langle c1, Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*blast dest*: *Seq-NoFaultStuckD1*)
next
fix c'
assume *steps-c'*: $\Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
assume *red*: $Seq\ c1\ c2 \in redexes\ c'$
from *redexes-subset* [*OF red*] *steps-c'*
show $\exists c'.\ \Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z) \wedge c1 \in redexes\ c'$
by (*auto iff*: *root-in-redexes*)

next
fix t
assume $\Gamma\vdash\langle Seq\ c1\ c2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c1,Normal\ Z\rangle \Rightarrow Normal\ t$
thus $\Gamma\vdash\langle c2,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*blast dest*: *Seq-NoFaultStuckD2*)
next
fix $c'\ t$
assume *steps-c'*: $\Gamma\vdash (Call\ p,\ Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ Z)$
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. 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⊢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 Cond.hyps by iprover
  have
  Γ,Θ⊢t/F ({s. s=Z ∧ Γ⊢⟨Cond b c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
∧
          Γ⊢Call p↓Normal σ ∧
          (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal s) ∧
                Cond b c1 c2 ∈ redexes c')}
          ∩ b)
          c1
         {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t},
         {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t}
  proof (rule ConseqMGT [OF hyp-c1],safe)
    assume Z ∈ b Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
    thus Γ⊢⟨c1,Normal Z⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
      by (auto simp add: final-notin-def intro: exec.CondTrue)
  next
    fix c'
    assume b: Z ∈ b
    assume steps-c': Γ⊢ (Call p, Normal σ) →+ (c', Normal Z)
    assume redex-c': Cond b c1 c2 ∈ redexes c'
    show ∃ c'. Γ⊢ (Call p, Normal σ) →+ (c', Normal Z) ∧ c1 ∈ redexes c'
    proof −
      note steps-c'
      also
      from b
      have Γ⊢(Cond b c1 c2, Normal Z) → (c1, Normal Z)
        by (rule step.CondTrue)
      from step-redexes [OF this redex-c'] obtain c'' where
        step-c'': Γ⊢ (c', Normal Z) → (c'', Normal Z) and
        c1: c1 ∈ 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*:
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F)) \wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \wedge c2 \in redexes\ c')\}$
$c2$
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *Cond.hyps* **by** *iprover*
have
$\Gamma,\Theta\vdash_{t/F} (\{s.\ s=Z \wedge \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))$
$\wedge$
$\Gamma\vdash Call\ p\downarrow Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',\ Normal\ s) \wedge$
$Cond\ b\ c1\ c2 \in redexes\ c')\}$
$\cap -b)$
$c2$
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c2*],*safe*)
assume $Z \notin b$ $\Gamma\vdash\langle Cond\ b\ c1\ c2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))$
thus $\Gamma\vdash\langle c2,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ \text{`}\ (-F))$
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 ∉ b Γ⊢⟨c2,Normal Z⟩ ⇒ Normal t
    thus Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t
      by (blast intro: exec.CondFalse)
  next
    fix t assume Z ∉ b Γ⊢⟨c2,Normal Z⟩ ⇒ Abrupt t
    thus Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t
      by (blast intro: exec.CondFalse)
  qed
  ultimately
  show
   Γ,Θ⊢t/F {s. s=Z ∧ Γ⊢⟨Cond b c1 c2,Normal s⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F))
∧
             Γ⊢Call p↓Normal σ ∧
           (∃ c'. Γ⊢(Call p,Normal σ) →+ (c',Normal s) ∧
                 Cond b c1 c2 ∈ redexes c')}
           (Cond b c1 c2)
          {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Normal t},
          {t. Γ⊢⟨Cond b c1 c2,Normal Z⟩ ⇒ Abrupt t}
    by (rule hoaret.Cond)
next
  case (While b c)
  let ?unroll = ({(s,t). s∈b ∧ Γ⊢⟨c,Normal s⟩ ⇒ Normal t})*
  let ?P' = λZ. {t. (Z,t)∈?unroll ∧
                  (∀ e. (Z,e)∈?unroll ⟶ e∈b
                       ⟶ Γ⊢⟨c,Normal e⟩ ⇒∉({Stuck} ∪ Fault ‘ (−F)) ∧
                          (∀ u. Γ⊢⟨c,Normal e⟩ ⇒Abrupt u ⟶
                               Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt u)) ∧
                  Γ⊢Call p↓Normal σ ∧
                (∃ c'. Γ⊢(Call p,Normal σ) →+
                           (c',Normal t) ∧ While b c ∈ redexes c')}
  let ?A = λZ. {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t}
  let ?r = {(t,s). Γ⊢(While b c)↓Normal s ∧ s∈b ∧
                  Γ⊢⟨c,Normal s⟩ ⇒ Normal t}
  show Γ,Θ⊢t/F
       {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')}
        (While b c)
       {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Normal t},
       {t. Γ⊢⟨While b c,Normal Z⟩ ⇒ Abrupt t}
  proof (rule ConseqMGT [where ?P'=λ Z. ?P' Z
                         and ?Q'=λ Z. ?P' Z ∩ − b])
    have wf-r: wf ?r by (rule wf-terminates-while)
    show ∀ Z. Γ,Θ⊢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*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s=Z \wedge \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \to^{+} (c',Normal\ s) \wedge c \in redexes\ c')\}$
c
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$ **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*: $s\in \{\tau\} \cap$
$\{t.\ (Z,\ t) \in ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}\ Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \to^{+}$
$(c',Normal\ t) \wedge While\ b\ c \in redexes\ c')\}$
$\cap\ b$
then obtain c' **where**
s-eq-τ: $s=\tau$ **and**
Z-s-unroll: $(Z,s) \in ?unroll$ **and**
noabort: $\forall e.\ (Z,e)\in ?unroll \longrightarrow e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)$ **and**
termi: $\Gamma\vdash Call\ p \downarrow Normal\ \sigma$ **and**
reach: $\Gamma\vdash(Call\ p,Normal\ \sigma) \to^{+} (c',Normal\ s)$ **and**
red-c': $While\ b\ c \in redexes\ c'$ **and**
s-in-b: $s\in b$
by *blast*
obtain c'' **where**
reach-c: $\Gamma\vdash(Call\ p,Normal\ \sigma) \to^{+} (c'',Normal\ s)$
$Seq\ c\ (While\ b\ c) \in redexes\ c''$
proof −
note *reach*
also from *s-in-b*
have $\Gamma\vdash(While\ b\ c,Normal\ s) \to (Seq\ c\ (While\ b\ c),Normal\ s)$
by (*rule step.WhileTrue*)
from *step-redexes* [*OF this red-c'*] **obtain** c'' **where**
step: $\Gamma\vdash (c',\ Normal\ s) \to (c'',\ Normal\ s)$ **and**
red-c'': $Seq\ c\ (While\ b\ c) \in redexes\ c''$
by *blast*
note *step*
finally

show *?thesis*
using *red-c''*
by (*blast intro*: *that*)
qed
from *reach termi*
have $\Gamma\vdash c' \downarrow$ *Normal s*
by (*rule steps-preserves-termination'*)
from *redexes-preserves-termination* [*OF this red-c'*]
have *termi-while*: $\Gamma\vdash$ *While b c* $\downarrow$ *Normal s* **.**
show $s \in \{t.\ t = s \wedge \Gamma\vdash\langle c,Normal\ t\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ t) \wedge c \in redexes\ c')\} \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Normal\ t\} \longrightarrow$
$t \in \{t.\ (t,\tau) \in ?r\} \cap$
$\{t.\ (Z,\ t) \in ?unroll \wedge$
$(\forall e.\ (Z,e)\in ?unroll \longrightarrow\ e\in b$
$\longrightarrow \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)) \wedge$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ t) \wedge$
$While\ b\ c \in redexes\ c')\}) \wedge$
$(\forall t.\ t \in \{t.\ \Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow Abrupt\ t\} \longrightarrow$
$t \in \{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ t\})$
(**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 $\Gamma\vdash$ (*c*, *Normal s*) $\rightarrow^*$ (*Skip*, *Normal t*).
  hence $\Gamma\vdash$ (*Seq c* (*While b c*), *Normal s*) $\rightarrow^*$
    (*Seq Skip* (*While b c*), *Normal t*)
    by (*rule SeqSteps*) *auto*
  moreover
  have $\Gamma\vdash$(*Seq Skip* (*While b c*), *Normal t*)$\rightarrow$(*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: $\Gamma\vdash$ (c'', *Normal s*) $\rightarrow^+$ (c''', *Normal t*) **and**
  red-c'': *While b c* $\in$ *redexes* c'''
  by *blast*
note *step*
finally
show *?thesis*
  using *red-c''*
  by (*blast intro*: *that*)
qed
moreover note *noabort termi*
ultimately
have $(t,\tau) \in$ *?r* $\wedge$ (*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* σ $\wedge$
    ($\exists$ c'. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^+$ (c', *Normal t*) $\wedge$
      While b c $\in$ *redexes* c')
  by *iprover*
}
then show *?C2* **by** *blast*
next
{
  fix *t*
  assume *s-t*: $\Gamma\vdash\langle$*c*,*Normal s*$\rangle$ $\Rightarrow$ *Abrupt t*
  from *Z-s-unroll noabort s-t s-in-b*
  have $\Gamma\vdash\langle$*While b c*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*
    by *blast*
} **thus** *?C3* **by** *simp*
qed
qed

qed
next
fix *s*
assume *P*: $s \in \{s.\ s{=}Z \wedge \Gamma\vdash\langle While\ b\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \wedge$
$While\ b\ c \in redexes\ c')\}$
hence *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by *auto*
show $s \in ?P'\ s \wedge$
$(\forall\, t.\ t{\in}(?P'\ s \cap -\ b){\longrightarrow}$
$t{\in}\{t.\ \Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Normal\ t\}){\wedge}$
$(\forall\, t.\ t{\in}?A\ s \longrightarrow t{\in}?A\ Z)$
proof (*intro conjI*)
{
fix *e*
assume $(Z,e) \in ?unroll\ e \in b$
from *this WhileNoFault*
have $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$(\forall\, u.\ \Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u \longrightarrow$
$\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow Abrupt\ u)$ (**is** *?Prop Z e*)
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
assume *e-in-b*: $e \in b$
assume *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
with *e-in-b WhileNoFault*
have *cNoFault*: $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
moreover
{
fix *u* **assume** $\Gamma\vdash\langle c,Normal\ e\rangle \Rightarrow Abrupt\ u$
with *e-in-b* **have** $\Gamma\vdash\langle While\ b\ c,Normal\ e\rangle \Rightarrow Abrupt\ u$
by (*blast intro*: *exec.intros*)
}
ultimately
show *?Prop e e*
by *iprover*
next
fix *Z r*
assume *e-in-b*: $e{\in}b$
assume *WhileNoFault*: $\Gamma\vdash\langle While\ b\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
assume *hyp*: $[\![e{\in}b;\Gamma\vdash\langle While\ b\ c,Normal\ r\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))]\!]$
$\Longrightarrow ?Prop\ r\ e$
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: $\Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ \text{`}\ (-F))$ **and**
Abrupt-r: $\forall u.\ \Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u \longrightarrow$
$\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ r\rangle \Rightarrow \mathit{Abrupt}\ u$
by *simp*

{
fix *u* **assume** $\Gamma\vdash\langle c,\mathit{Normal}\ e\rangle \Rightarrow \mathit{Abrupt}\ u$
with *Abrupt-r* **have** $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ r\rangle \Rightarrow \mathit{Abrupt}\ u$ **by** *simp*
moreover from *Z-r* **obtain**
$Z \in b$ $\Gamma\vdash\langle c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ r$
by *simp*
ultimately have $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ u$
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 $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t$
proof (*induct rule*: *converse-rtrancl-induct* [*consumes 1*])
from *termination*
show $\Gamma\vdash\langle \mathit{While}\ b\ c,\mathit{Normal}\ t\rangle \Rightarrow \mathit{Normal}\ t$
by (*blast intro*: *exec.WhileFalse*)
next
fix *Z r*
assume *first-body*:
$(Z,\ r) \in \{(s,\ t).\ s \in b \wedge \Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow \mathit{Normal}\ t\}$
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 $\forall t.\ t \in (?P'\ s \cap -\ b)$
$\longrightarrow t \in \{t.\ \Gamma\vdash\langle While\ b\ c, Normal\ Z\rangle \Rightarrow Normal\ t\}$
by *blast*
next
from P **show** $\forall t.\ t \in ?A\ s \longrightarrow t \in ?A\ Z$
by *simp*
qed
qed
next
case (*Call q*)
let $?P = \{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q\ , Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ s) \wedge Call\ q \in redexes\ c')\}$
from *noStuck-Call*
have $\forall s \in ?P.\ q \in dom\ \Gamma$
by (*fastforce simp add*: *final-notin-def*)
then show *?case*
proof (*rule conseq-extract-state-indep-prop*)
assume *q-defined*: $q \in dom\ \Gamma$
from *Call-hyp* **have**
$\forall q \in dom\ \Gamma.\ \forall Z.$
$\Gamma,\Theta\vdash_{t/F}\{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ q{\downarrow}Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*simp add*: *exec-Call-body'* *noFaultStuck-Call-body'* [*simplified*]
terminates-Normal-Call-body)
from *Call-hyp q-defined* **have** *Call-hyp'*:
$\forall Z.\ \Gamma,\Theta \vdash_{t/F} \{s.\ s=Z \wedge \Gamma\vdash\langle Call\ q, Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\wedge$
$\Gamma\vdash Call\ q{\downarrow}Normal\ s \wedge ((s,q),(\sigma,p)) \in termi\text{-}call\text{-}steps\ \Gamma\}$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q, Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by *auto*
show
$\Gamma,\Theta\vdash_{t/F}\ ?P$
$(Call\ q)$
$\{t.\ \Gamma\vdash\langle Call\ q\ , Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Call\ q\ , Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF Call-hyp'*], *safe*)
fix c'
assume *termi*: $\Gamma\vdash Call\ p \downarrow Normal\ \sigma$
assume *steps-c'*: $\Gamma\vdash (Call\ p, Normal\ \sigma) \rightarrow^{+} (c', Normal\ Z)$

assume *red-c′*: *Call q* $\in$ *redexes c′*
show $\Gamma\vdash$*Call q* $\downarrow$ *Normal Z*
proof $-$
from *steps-preserves-termination′* [*OF steps-c′ termi*]
have $\Gamma\vdash c' \downarrow$ *Normal Z* **.**
from *redexes-preserves-termination* [*OF this red-c′*]
show *?thesis* **.**
qed
next
fix *c′*
assume *termi*: $\Gamma\vdash$*Call p* $\downarrow$ *Normal* σ
assume *steps-c′*: $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^{+}$ (c', *Normal Z*)
assume *red-c′*: *Call q* $\in$ *redexes c′*
from *redex-redexes* [*OF this*]
have *redex c′* = *Call q*
by *auto*
with *termi steps-c′*
show ((*Z*, *q*), σ, *p*) $\in$ *termi-call-steps* Γ
by (*auto simp add*: *termi-call-steps-def*)
qed
qed
next
case (*DynCom c*)
have *hyp*:
$\bigwedge s'. \forall Z. \Gamma,\Theta\vdash_{t/F}$
$\{s.\ s = Z \land \Gamma\vdash\langle c\ s',Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \land$
$(\exists c'. \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land c\ s' \in redexes\ c')\}$
$(c\ s')$
$\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle c\ s',Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
using *DynCom* **by** *simp*
have *hyp′*:
$\Gamma,\Theta\vdash_{t/F} \{s.\ s{=}Z \land \Gamma\vdash\langle DynCom\ c,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \land$
$\Gamma\vdash Call\ p \downarrow Normal\ \sigma \land$
$(\exists c'. \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^{+} (c',Normal\ s) \land DynCom\ c \in redexes$
$c')\}$
$(c\ Z)$
$\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow Normal\ t\},\{t.\ \Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow$
$Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp*],*safe*)
assume $\Gamma\vdash\langle DynCom\ c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
then show $\Gamma\vdash\langle c\ Z,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*fastforce simp add*: *final-notin-def intro*: *exec.intros*)
next
fix *c′*
assume *steps*: $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^{+}$ (c', *Normal Z*)
assume *c′*: *DynCom c* $\in$ *redexes c′*
have $\Gamma\vdash$ (*DynCom c*, *Normal Z*) $\rightarrow$ (*c Z*,*Normal Z*)
by (*rule step.DynCom*)

from *step-redexes* [*OF this c′*] **obtain** c'' **where**
step: $\Gamma\vdash$ (c', *Normal Z*) $\rightarrow$ (c'', *Normal Z*) **and** c'': *c Z* $\in$ *redexes* c''
by *blast*
note *steps* **also note** *step*
finally show $\exists\, c'$. $\Gamma\vdash$ (*Call p*, *Normal* σ) $\rightarrow^{+}$ (c', *Normal Z*) $\wedge$ *c Z* $\in$ *redexes*
c'
using c'' **by** *blast*
next
fix *t*
assume $\Gamma\vdash\langle$*c Z*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*
thus $\Gamma\vdash\langle$*DynCom c*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*
by (*auto intro*: *exec.intros*)
next
fix *t*
assume $\Gamma\vdash\langle$*c Z*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*
thus $\Gamma\vdash\langle$*DynCom c*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*
by (*auto intro*: *exec.intros*)
qed
show *?case*
apply (*rule hoaret.DynCom*)
apply *safe*
apply (*rule hyp′*)
done
next
case (*Guard f g c*)
have *hyp-c*: $\forall\, Z$. $\Gamma,\Theta\vdash_{t/F}$
{*s*. *s*=*Z* $\wedge$ $\Gamma\vdash\langle$*c*,*Normal s*$\rangle$ $\Rightarrow\notin$({*Stuck*} $\cup$ *Fault* ‘ ($-F$)) $\wedge$
$\Gamma\vdash$*Call p*$\downarrow$*Normal* σ $\wedge$
($\exists\, c'$. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^{+}$ (c',*Normal s*) $\wedge$ *c* $\in$ *redexes* c')}
c
{*t*. $\Gamma\vdash\langle$*c*,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*},
{*t*. $\Gamma\vdash\langle$*c*,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*}
using *Guard.hyps* **by** *iprover*
show $\Gamma,\Theta\vdash_{t/F}$ {*s*. *s*=*Z* $\wedge$ $\Gamma\vdash\langle$*Guard f g c* ,*Normal s*$\rangle$ $\Rightarrow\notin$({*Stuck*} $\cup$ *Fault* ‘
($-F$)) $\wedge$
$\Gamma\vdash$*Call p*$\downarrow$*Normal* σ $\wedge$
($\exists\, c'$. $\Gamma\vdash$(*Call p*,*Normal* σ) $\rightarrow^{+}$ (c',*Normal s*) $\wedge$ *Guard f g c* $\in$ *redexes*
c')}
Guard f g c
{*t*. $\Gamma\vdash\langle$*Guard f g c* ,*Normal Z*$\rangle$ $\Rightarrow$ *Normal t*},
{*t*. $\Gamma\vdash\langle$*Guard f g c* ,*Normal Z*$\rangle$ $\Rightarrow$ *Abrupt t*}
proof (*cases f* $\in$ *F*)
case *True*
have $\Gamma,\Theta\vdash_{t/F}$ (*g* $\cap$ {*s*. *s*=*Z* $\wedge$
$\Gamma\vdash\langle$*Guard f g c* ,*Normal s*$\rangle$ $\Rightarrow\notin$({*Stuck*} $\cup$ *Fault* ‘ ($-F$)) $\wedge$
$\Gamma\vdash$*Call p*$\downarrow$*Normal* 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 $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))\ Z\in g$
thus $\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z)$
assume c': $Guard\ f\ g\ c \in redexes\ c'$
assume $Z \in g$
from *this* **have** $\Gamma\vdash(Guard\ f\ g\ c,Normal\ Z) \rightarrow (c,Normal\ Z)$
by (*rule step.Guard*)
from *step-redexes* [*OF this* c'] **obtain** c'' **where**
step: $\Gamma\vdash\ (c',\ Normal\ Z) \rightarrow (c'',\ Normal\ Z)$ **and** c'': $c \in redexes\ c''$
by *blast*
note *steps* **also note** *step*
finally show $\exists\, c'.\ \Gamma\vdash\ (Call\ p,\ Normal\ \sigma) \rightarrow^+ (c',\ Normal\ Z) \wedge c \in redexes$
c'
using c'' **by** *blast*
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Normal\ t\ Z \in g$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix t
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
$\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow Abrupt\ t\ Z \in g$
thus $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
qed
from *True this* **show** *?thesis*
by (*rule conseqPre* [*OF Guarantee*]) *auto*
next
case *False*
have $\Gamma,\Theta\vdash_{t/F} (g \cap \{s.\ s{=}Z \wedge$
$\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(Call\ p,Normal\ \sigma) \rightarrow^+ (c',Normal\ s) \wedge$
$Guard\ f\ g\ c \in redexes\ c')\})$
c
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Normal\ t\}$,
$\{t.\ \Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof (*rule ConseqMGT* [*OF hyp-c*], *safe*)
assume $\Gamma\vdash\langle Guard\ f\ g\ c\ ,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
thus $\Gamma\vdash\langle c,Normal\ Z\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F))$
using *False*

by (*cases* $Z\in g$) (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c'
assume *steps*: $\Gamma\vdash$ (*Call p*, *Normal* 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*
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* σ $\wedge$
$(\exists\, c'$. $\Gamma\vdash$(*Call p*, *Normal* σ) $\rightarrow^+$ (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*])

($\mathit{blast\ intro}$: $\mathit{exec.intros\ terminates.intros}$)

next

case ($\mathit{Catch}\ c_1\ c_2$)

have $\mathit{hyp\text{-}c1}$:

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

have $\mathit{hyp\text{-}c2}$:

$\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ \{s.\ s= Z \wedge \Gamma\vdash\langle c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ p\downarrow \mathit{Normal}\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^+ (c',\mathit{Normal}\ s) \wedge c_2 \in \mathit{redexes}\ c')\}$
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 $\mathit{Catch.hyps}$ **by** $\mathit{iprover}$

have

$\Gamma,\Theta\vdash_{t/F}\ \{s.\ s = Z \wedge \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash \mathit{Call}\ p\downarrow \mathit{Normal}\ \sigma \wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma)\rightarrow^+(c',\mathit{Normal}\ s) \wedge$
$\mathit{Catch}\ c_1\ c_2 \in \mathit{redexes}\ c')\}$
c_1
$\{t.\ \Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}`(-F)) \wedge \Gamma\vdash \mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma$
$\wedge$
$(\exists c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^+ (c',\mathit{Normal}\ t) \wedge c_2 \in \mathit{redexes}\ c')\}$

proof ($\mathit{rule\ ConseqMGT}$ [$\mathit{OF\ hyp\text{-}c1}$],$\mathit{clarify}$,$\mathit{safe}$)

assume $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$

thus $\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$

by ($\mathit{fastforce\ simp\ add}$: $\mathit{final\text{-}notin\text{-}def\ intro}$: $\mathit{exec.intros}$)

next

fix c'

assume steps: $\Gamma\vdash\ (\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \rightarrow^+ (c',\ \mathit{Normal}\ Z)$

assume c': $\mathit{Catch}\ c_1\ c_2 \in \mathit{redexes}\ c'$

from $\mathit{steps\ redexes\text{-}subset}$ [$\mathit{OF\ this}$]

show $\exists c'.\ \Gamma\vdash\ (\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \rightarrow^+ (c',\ \mathit{Normal}\ Z) \wedge c_1 \in \mathit{redexes}\ c'$

by ($\mathit{auto\ iff}$: $\mathit{root\text{-}in\text{-}redexes}$)

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 ($\mathit{auto\ intro}$: $\mathit{exec.intros}$)

next

fix t

assume $\Gamma\vdash\langle \mathit{Catch}\ c_1\ c_2,\mathit{Normal}\ Z\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$

$\Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
thus $\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
by (*auto simp add*: *final-notin-def intro*: *exec.intros*)
next
fix c' t
assume *steps-c'*: $\Gamma\vdash$ $(\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \rightarrow^+ (c',\ \mathit{Normal}\ Z)$
assume *red*: $\mathit{Catch}\ c_1\ c_2 \in \mathit{redexes}\ c'$
assume *exec-c*$_1$: $\Gamma\vdash$ $\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t$
show $\exists\, c'.\ \Gamma\vdash$ $(\mathit{Call}\ p,\ \mathit{Normal}\ \sigma) \rightarrow^+ (c',\ \mathit{Normal}\ t) \wedge c_2 \in \mathit{redexes}\ c'$
proof $-$
note *steps-c'*
also
from *exec-impl-steps-Normal-Abrupt* [*OF exec-c*$_1$]
have $\Gamma\vdash$ $(c_1,\ \mathit{Normal}\ Z) \rightarrow^* (\mathit{Throw},\ \mathit{Normal}\ t)$.
from *steps-redexes-Catch* [*OF this red*]
obtain c'' **where**
steps-c'': $\Gamma\vdash$ $(c',\ \mathit{Normal}\ Z) \rightarrow^* (c'',\ \mathit{Normal}\ t)$ **and**
Catch: $\mathit{Catch}\ \mathit{Throw}\ c_2 \in \mathit{redexes}\ c''$
by *blast*
note *steps-c''*
also
have *step-Catch*: $\Gamma\vdash$ $(\mathit{Catch}\ \mathit{Throw}\ c_2,\mathit{Normal}\ t) \rightarrow (c_2,\mathit{Normal}\ t)$
by (*rule step.CatchThrow*)
from *step-redexes* [*OF step-Catch Catch*]
obtain c''' **where**
step-c''': $\Gamma\vdash$ $(c'',\ \mathit{Normal}\ t) \rightarrow (c''',\ \mathit{Normal}\ t)$ **and**
c2: $c_2 \in \mathit{redexes}\ c'''$
by *blast*
note *step-c'''*
finally show *?thesis*
using *c2*
by *blast*
qed
qed
moreover
have $\Gamma,\Theta\vdash_{t/F} \{t.\ \Gamma\vdash\langle c_1,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\ \wedge$
$\Gamma\vdash\langle c_2,\mathit{Normal}\ t\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash\mathit{Call}\ p \downarrow \mathit{Normal}\ \sigma\ \wedge$
$(\exists\, c'.\ \Gamma\vdash(\mathit{Call}\ p,\mathit{Normal}\ \sigma) \rightarrow^+ (c',\mathit{Normal}\ t) \wedge c_2 \in \mathit{redexes}\ c')\}$
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 *A*:
$\forall\, q \in \mathit{dom}\ \Gamma.\ \forall\, Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ q\downarrow \mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
assumes *pdef*: $p \in \mathit{dom}\ \Gamma$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$(\{\sigma\} \cap \{s.\ s{=}Z \wedge\Gamma\vdash\langle \mathit{the}\ (\Gamma\ p),\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash \mathit{the}\ (\Gamma\ p)\downarrow \mathit{Normal}\ s\})$
$\mathit{the}\ (\Gamma\ p)$
$\{t.\ \Gamma\vdash\langle \mathit{the}\ (\Gamma\ p),\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{the}\ (\Gamma\ p),\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
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: $\forall\, q \in \mathit{dom}\ \Gamma.\ \forall\, Z.\ \Gamma,\Theta \vdash_{t/F}$
$\{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F)) \wedge$
$\Gamma\vdash \mathit{Call}\ q\downarrow \mathit{Normal}\ s \wedge ((s,q),(\sigma,p)) \in \mathit{termi\text{-}call\text{-}steps}\ \Gamma\}$
$(\mathit{Call}\ q)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ q,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
assumes *p-defined*: $p \in \mathit{dom}\ \Gamma$
shows $\bigwedge Z.\ \Gamma,\Theta \vdash_{t/F}$
$(\{\sigma\} \cap \{s.\ s{=}Z \wedge \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ s\rangle \Rightarrow\notin(\{\mathit{Stuck}\} \cup \mathit{Fault}\ `\ (-F))$
$\wedge$
$\Gamma\vdash \mathit{Call}\ p\downarrow \mathit{Normal}\ s\})$
$\mathit{the}\ (\Gamma\ p)$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Normal}\ t\},$
$\{t.\ \Gamma\vdash\langle \mathit{Call}\ p,\mathit{Normal}\ Z\rangle \Rightarrow \mathit{Abrupt}\ t\}$
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** $Procs{=}dom\ \Gamma$ **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**
$r{=}termi\text{-}call\text{-}steps\ \Gamma$
])
apply *simp*
apply *simp*
apply (*rule wf-termi-call-steps*)
apply (*intro ballI allI*)
apply *simp*
apply (*rule Call-lemma-switch-Call-body* [*rule-format*, *simplified*])
apply (*rule hoaret.Asm*)
apply *fastforce*
apply *assumption*
done

lemma *CollInt-iff*: $\{s.\ P\ s\} \cap \{s.\ Q\ s\} = \{s.\ P\ s \wedge Q\ s\}$
by *auto*

lemma *image-Un-conv*: $f\ `\ (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.\ \{x\ p\ Z\}) = (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.\ \{f\ (x\ p\ Z)\})$
by (*auto iff*: *not-None-eq*)

Another proof of *MGT-Call*, maybe a little more readable

lemma
$\forall p \in dom\ \Gamma.\ \forall Z.$
$\Gamma,\{\} \vdash_{t/F} \{s.\ s{=}Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash(Call\ p)\downarrow Normal\ s\}$
$(Call\ p)$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
proof −
{
fix $p\ Z\ \sigma$
assume *defined*: $p \in dom\ \Gamma$

define *Specs* **where** $Specs = (\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s{=}Z\ \wedge$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ s\},$
$p,$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\})\})$
define *Specs-wf* **where** $Specs\text{-}wf\ p\ \sigma = (\lambda(P,q,Q,A).$
$(P \cap \{s.\ ((s,q),\sigma,p) \in termi\text{-}call\text{-}steps\ \Gamma\},\ q,\ Q,\ A))\ `\ Specs$ **for**
$p\ \sigma$
have $\Gamma,Specs\text{-}wf\ p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash the\ (\Gamma\ p){\downarrow}Normal\ s\})$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle the\ (\Gamma\ p),Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
apply (*rule Call-lemma* [*rule-format*, *OF - defined*])
apply (*rule hoaret.Asm*)
apply (*clarsimp simp add*: *Specs-wf-def Specs-def image-Un-conv*)
apply (*rule-tac x=q* **in** *bexI*)
apply (*rule-tac x=Z* **in** *exI*)
apply (*clarsimp simp add*: *CollInt-iff*)
apply *auto*
done
hence $\Gamma,Specs\text{-}wf\ p\ \sigma$
$\vdash_{t/F}(\{\sigma\} \cap$
$\{s.\ s = Z \wedge \Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ s\})$
$(the\ (\Gamma\ p))$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\}$
by (*simp only*: *exec-Call-body'* [*OF defined*]
noFaultStuck-Call-body' [*OF defined*]
terminates-Normal-Call-body [*OF defined*])
} **note** *bdy=this*
show *?thesis*
apply (*intro ballI allI*)
apply (*rule hoaret.CallRec* [**where** $Specs{=}(\bigcup p{\in}dom\ \Gamma.\ \bigcup Z.$
$\{(\{s.\ s{=}Z\ \wedge$
$\Gamma\vdash\langle Call\ p,Normal\ s\rangle \Rightarrow\notin(\{Stuck\} \cup Fault\ `\ (-F)) \wedge$
$\Gamma\vdash Call\ p{\downarrow}Normal\ s\},$
$p,$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Normal\ t\},$
$\{t.\ \Gamma\vdash\langle Call\ p,Normal\ Z\rangle \Rightarrow Abrupt\ t\})\}),$
OF - wf-termi-call-steps [*of* Γ] *refl*])
apply *fastforce*
apply *clarify*
apply (*rule conjI*)

apply *fastforce*
apply (*rule allI*)
apply (*simp* (*no-asm-use*) *only* : *Un-empty-left*)
apply (*rule bdy*)
apply *auto*
done
qed

theorem *hoaret-complete*: $\Gamma \models_{t/F} P\ c\ Q,A \Longrightarrow \Gamma,\{\} \vdash_{t/F} P\ c\ Q,A$
by (*iprover intro*: *MGT-implies-complete MGT-lemma* [*OF MGT-Call*])

lemma *hoaret-complete'*:
assumes *cvalid*: $\Gamma,\Theta \models_{t/F} P\ c\ Q,A$
shows $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
proof (*cases* $\Gamma \models_{t/F} P\ c\ Q,A$)
case *True*
hence $\Gamma,\{\} \vdash_{t/F} P\ c\ Q,A$
by (*rule hoaret-complete*)
thus $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
by (*rule hoaret-augment-context*) *simp*
next
case *False*
with *cvalid*
show *?thesis*
by (*rule ExFalso*)
qed

13.3 And Now: Some Useful Rules

13.3.1 Modify Return

lemma *ProcModifyReturn-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F} P$ *call init p return' c* Q,A
assumes *valid-modif*:
$\forall \sigma.\ \Gamma,\Theta \models_{/UNIV} \{\sigma\}$ (*Call p*) (*Modif* 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*: $\Gamma\ p = Some\ bdy$
  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 Γ⊢⟨call init p return' c ,Normal s⟩ ⇒ t
      by (auto intro: exec-callFault)
    from cvalidt-postD [OF valid-call ctxt this P] t t-notin-F
    show ?thesis
      by simp
  next
    fix bdy
    assume bdy: Γ p = Some bdy
    assume 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 ⊨t/F P (Call p) Q,A
  hence ∀ (P, p, Q, A)∈Θ. Γ ⊨/F P (Call p) Q,A
    by (auto simp add: validt-def)
  then have ctxt': ∀ (P, p, Q, A)∈Θ. Γ ⊨/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*:
$\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ (Call\ p)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta\vdash_{t/F}$ *P* (*call init p return c*) *Q*,*A*
apply (*rule hoaret-complete'*)
apply (*rule ProcModifyReturnSameFaults-sound* [**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,
OF - - *res-modif ret-modifAbr*])
apply (*rule hoaret-sound* [*OF spec*])
using *modifies-spec*
apply (*blast intro*: *hoare-sound*)
done

13.3.2 DynCall

lemma *dynProcModifyReturn-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F}$ *P dynCall init p return' c Q*,*A*
assumes *valid-modif*:
$\forall s\in P.\ \forall \sigma.\ \Gamma,\Theta \models_{/UNIV} \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),(ModifAbr\ \sigma)$
assumes *ret-modif*:
$\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$
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*: $\Gamma\vdash call\ init\ (p\ s)\ return'\ c\downarrow\ Normal\ s$
  by *cases*
have $\Gamma\vdash call\ init\ (p\ s)\ return\ c\ \downarrow\ Normal\ s$
proof (*cases* $p\ s \in dom\ \Gamma$)
  case *True*
  with *call* **obtain** *bdy* **where**
    bdy: $\Gamma\ (p\ s) = Some\ bdy$ **and** *termi-bdy*: $\Gamma\vdash bdy\ \downarrow\ Normal\ (init\ s)$ **and**
    termi-c: $\forall t.\ \Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t \longrightarrow$
        $\Gamma\vdash c\ s\ t\ \downarrow\ Normal\ (return'\ s\ t)$
    by *cases auto*
  {
    fix *t*
    assume *exec-bdy*: $\Gamma\vdash\langle bdy,Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
    hence $\Gamma\vdash c\ s\ t\ \downarrow\ Normal\ (return\ s\ t)$
    proof −
      from *exec-bdy bdy*
      have $\Gamma\vdash\langle Call\ (p\ s),Normal\ (init\ s)\rangle \Rightarrow Normal\ t$
        by (*auto simp add*: *intro*: *exec.intros*)
      from *cvalidD* [*OF valid-modif* [*rule-format*, *of s init s*] *ctxt' this*] *P*
        ret-modif
      have $return'\ s\ t = return\ s\ t$
        by *auto*
      with *termi-c exec-bdy* **show** *?thesis* **by** *auto*
    qed
  }
  with *bdy termi-bdy*
  show *?thesis*
    by (*iprover intro*: *terminates-call*)
next
  case *False*
  thus *?thesis*
    by (*auto intro*: *terminates-callUndefined*)
qed
thus $\Gamma\vdash dynCall\ init\ p\ return\ c\ \downarrow\ Normal\ s$
  by (*iprover intro*: *terminates-dynCall*)
qed

lemma *dynProcModifyReturn*:
assumes *dyn-call*: $\Gamma,\Theta\vdash_{t/F} P\ dynCall\ init\ p\ return'\ c\ Q,A$
assumes *ret-modif*:
  $\forall s\ t.\ t \in Modif\ (init\ s)$

$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *ret-modifAbr*: $\forall s\ t.\ t \in ModifAbr\ (init\ s)$
$\longrightarrow return'\ s\ t = return\ s\ t$
assumes *modif*:
$\forall s \in P.\ \forall \sigma.$
$\Gamma,\Theta \vdash_{/UNIV} \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule dynProcModifyReturn-sound*
[**where** *Modif=Modif* **and** *ModifAbr=ModifAbr*,
OF hoaret-sound [*OF dyn-call*] - *ret-modif ret-modifAbr*])
apply (*intro ballI allI*)
apply (*rule hoare-sound* [*OF modif* [*rule-format*]])
apply *assumption*
done

lemma *dynProcModifyReturnSameFaults-sound*:
assumes *valid-call*: $\Gamma,\Theta \models_{t/F} P\ dynCall\ init\ p\ return'\ c\ Q,A$
assumes *valid-modif*:
$\forall s{\in}P.\ \forall \sigma.\ \Gamma,\Theta \models_{/F} \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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 $\Gamma,\Theta \models_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
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'*:
$\forall \sigma.\ \Gamma,\Theta\models_{/F} \{\sigma\}\ (Call\ (p\ s))\ (Modif\ \sigma),(ModifAbr\ \sigma)$
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.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 Γ⊢⟨*call init* (*p s*) *return*′ *c*,*Normal s*⟩ ⇒ *t*
  by (*auto intro*: *exec-call*)
hence Γ⊢⟨*dynCall init p return*′ *c*,*Normal s*⟩ ⇒ *t*
  by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy t*′
assume *bdy*: Γ (*p s*) = *Some bdy*
assume *exec-body*: Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Abrupt t*′
assume *t*: *t* = *Abrupt* (*return s t*′)
also from *exec-body bdy*
have Γ⊢⟨*Call* (*p s*) ,*Normal* (*init s*)⟩ ⇒ *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 Γ⊢⟨*call init* (*p s*) *return*′ *c*,*Normal s*⟩ ⇒ *t*
  by (*auto intro*: *exec-callAbrupt*)
hence Γ⊢⟨*dynCall init p return*′ *c*,*Normal s*⟩ ⇒ *t*
  by (*rule exec-dynCall*)
from *cvalidt-postD* [*OF valid-call ctxt this*] *P t-notin-F*
show *?thesis*
  by *simp*
next
fix *bdy f*
assume *bdy*: Γ (*p s*) = *Some bdy*
assume Γ⊢⟨*bdy*,*Normal* (*init s*)⟩ ⇒ *Fault f* **and**
  t: *t* = *Fault f*
with *bdy* **have** Γ⊢⟨*call init* (*p s*) *return*′ *c* ,*Normal s*⟩ ⇒ *t*
  by (*auto intro*: *exec-callFault*)
hence Γ⊢⟨*dynCall init p return*′ *c*,*Normal s*⟩ ⇒ *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*: Γ $(p\ s) = Some\ bdy$
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* $p\ s \in dom\ \Gamma$)
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

13.3.3 Conjunction of Postcondition

lemma *PostConjI-sound*:
assumes *valid-Q*: $\Gamma,\Theta \models_{t/F}$ *P c Q*,*A*
assumes *valid-R*: $\Gamma,\Theta \models_{t/F}$ *P c R*,*B*
shows $\Gamma,\Theta \models_{t/F}$ *P c* ($Q \cap R$),($A \cap B$)
proof (*rule cvalidtI*)
fix *s t*
assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma \models_{t/F} P$ (*Call p*) *Q*,*A*
assume *exec*: $\Gamma\vdash\langle c,$*Normal s*$\rangle \Rightarrow t$
assume *P*: $s \in P$
assume *t-notin-F*: $t \notin$ *Fault* ‘ *F*

from *valid-Q ctxt exec P t-notin-F* **have** $t \in Normal$ ' $Q \cup Abrupt$ ' A
  by (*rule cvalidt-postD*)
moreover
from *valid-R ctxt exec P t-notin-F* **have** $t \in Normal$ ' $R \cup Abrupt$ ' B
  by (*rule cvalidt-postD*)
ultimately show $t \in Normal$ ' $(Q \cap R) \cup Abrupt$ ' $(A \cap B)$
  by *blast*
next
fix s
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma \models_{t/F} P\ (Call\ p)\ Q,A$
assume P: $s \in P$
from *valid-Q ctxt P*
show $\Gamma\vdash c \downarrow Normal\ s$
  by (*rule cvalidt-termD*)
qed

lemma *PostConjI*:
assumes *deriv-Q*: $\Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
assumes *deriv-R*: $\Gamma,\Theta\vdash_{t/F} P\ c\ R,B$
shows $\Gamma,\Theta\vdash_{t/F} P\ c\ (Q \cap R),(A \cap B)$
apply (*rule hoaret-complete'*)
apply (*rule PostConjI-sound*)
apply (*rule hoaret-sound* [*OF deriv-Q*])
apply (*rule hoaret-sound* [*OF deriv-R*])
done

lemma *Merge-PostConj-sound*:
assumes *validF*: $\Gamma,\Theta\models_{t/F} P\ c\ Q,A$
assumes *validG*: $\Gamma,\Theta\models_{t/G} P'\ c\ R,X$
assumes *F-G*: $F \subseteq G$
assumes *P-P'*: $P \subseteq P'$
shows $\Gamma,\Theta\models_{t/F} P\ c\ (Q \cap R),(A \cap X)$
proof (*rule cvalidtI*)
fix $s\ t$
assume *ctxt*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/F} P\ (Call\ p)\ Q,A$
with *F-G* **have** *ctxt'*: $\forall (P, p, Q, A)\in\Theta.\ \Gamma\models_{t/G} P\ (Call\ p)\ Q,A$
  by (*auto intro*: *validt-augment-Faults*)
assume *exec*: $\Gamma\vdash\langle c,Normal\ s\rangle \Rightarrow t$
assume P: $s \in P$
with *P-P'* **have** P': $s \in P'$
  by *auto*
assume *t-noFault*: $t \notin Fault$ ' F
show $t \in Normal$ ' $(Q \cap R) \cup Abrupt$ ' $(A \cap X)$
proof −
  from *cvalidt-postD* [*OF validF* [*rule-format*] *ctxt exec P t-noFault*]
  have t: $t \in Normal$ ' $Q \cup Abrupt$ ' 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

13.3.4 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 Γ⊢⟨c₁,Normal s⟩ ⇒ 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 Γ⊢⟨c₁,Normal s⟩ ⇒ Fault f ∨ Γ⊢⟨c₂,Normal s⟩ ⇒ Fault f
      by auto
    then show ?thesis
    proof (cases rule: disjE [consumes 1])
      assume Γ⊢⟨c₁,Normal s⟩ ⇒ Fault f
      from cvalidt-postD [OF valid-c1 ctxt this P] t t-notin-F
      show ?thesis
        by blast
    next
      assume Γ⊢⟨c₂,Normal s⟩ ⇒ 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 Γ⊢⟨c₁,Normal s⟩ ⇒ t by simp
    from valid-c1 ctxt this P t-notin-F
    show ?thesis
      by (rule cvalidt-postD)
  qed
next
  fix s
  assume ctxt: ∀(P, p, Q, A)∈Θ. Γ ⊨t/F P (Call p) Q,A
  assume P: s ∈ P
  show Γ⊢c ↓ Normal s
  proof −
    from valid-c1 ctxt P
    have Γ⊢c₁ ↓ 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*: $\Gamma,\Theta \models_{/\{\}} P\ (strip\text{-}guards\ (-F)\ c)\ UNIV,UNIV$
  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 $\Gamma \vdash \langle strip\text{-}guards\ (-F)\ c, Normal\ s\rangle \Rightarrow Fault\ f$
        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: ∀ (P, p, Q, A)∈Θ. Γ ⊨t/{} P (Call p) Q,A
  hence ctxt': ∀ (P, p, Q, A)∈Θ. Γ⊨t/F P (Call p) Q,A
    by (auto intro: valid-augment-Faults simp add: validt-def)
  assume P: s ∈ P
  show Γ⊢c ↓ Normal s
  proof −
    from valid ctxt' P
    show Γ⊢c ↓ Normal s
      by (rule cvalidt-termD)
  qed
qed

lemma CombineStrip:
  assumes deriv: Γ,Θ⊢t/F P c Q,A
  assumes deriv-strip: Γ,Θ⊢/{} P (strip-guards (−F) c) UNIV,UNIV
  shows Γ,Θ⊢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: Γ,Θ⊨t/F P c Q,A
  assumes validFlip: Γ,Θ⊨/−F P c UNIV,UNIV
  shows Γ,Θ⊨t/{} P c Q,A
proof (rule cvalidtI)
  fix s t
  assume ctxt: ∀ (P, p, Q, A)∈Θ. Γ⊨t/{} P (Call p) Q,A
  from ctxt have ctxt': ∀ (P, p, Q, A)∈Θ. Γ⊨t/F P (Call p) Q,A
    by (auto intro: valid-augment-Faults simp add: validt-def)
  from ctxt have ctxtFlip: ∀ (P, p, Q, A)∈Θ. Γ⊨/−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** $\Gamma\vdash\langle mark\text{-}guards\ f\ c,Normal\ s\rangle \Rightarrow Fault\ f'$
      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** $\Gamma\vdash\langle mark\text{-}guards\ f\ c,Normal\ s\rangle \Rightarrow t$
      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 $\Gamma\vdash mark\text{-}guards\ f\ c \downarrow Normal\ s$**.**
  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 $\Gamma \vdash merge\text{-}guards\ c \downarrow Normal\ s$.
  thus $\Gamma \vdash c \downarrow Normal\ s$
    by (*rule terminates-merge-guards-to-terminates*)
qed

lemma *MergeGuardsD*:
  assumes *deriv*: $\Gamma,\Theta \vdash_{t/F} P\ merge\text{-}guards\ c\ Q,A$
  shows $\Gamma,\Theta \vdash_{t/F} P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule MergeGuardsD-sound*)
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *SubsetGuards-sound*:
  assumes *c-c'*: $c \subseteq_g c'$
  assumes *valid*: $\Gamma,\Theta \models_{t/\{\}} P\ c'\ Q,A$
  shows $\Gamma,\Theta \models_{t/\{\}} P\ c\ Q,A$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models_{t/\{\}} P\ (Call\ p)\ Q,A$
  assume *exec*: $\Gamma \vdash \langle c, Normal\ s\rangle \Rightarrow t$
  from *exec-to-exec-subseteq-guards* [*OF c-c' exec*] **obtain** t' **where**
    exec-c': $\Gamma \vdash \langle c', Normal\ s\rangle \Rightarrow t'$ **and**
    t'-noFault: $\neg\ isFault\ t' \longrightarrow t' = t$
    by *blast*
  assume *P*: $s \in P$
  assume *t-noFault*: $t \notin Fault$ ` $\{\}$
  from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-c' P*] *t'-noFault t-noFault*
  show $t \in Normal$ ` $Q \cup Abrupt$ ` A
    by *auto*
next
  fix *s*
  assume *ctxt*: $\forall (P, p, Q, A) \in \Theta.\ \Gamma \models_{t/\{\}} P\ (Call\ p)\ Q,A$
  assume *P*: $s \in P$
  from *cvalidt-termD* [*OF valid ctxt P*]
  have *termi-c'*: $\Gamma \vdash c' \downarrow Normal\ s$.
  from *cvalidt-postD* [*OF valid ctxt - P*]
  have *noFault-c'*: $\Gamma \vdash \langle c', Normal\ s\rangle \Rightarrow \notin Fault$ ` $UNIV$
    by (*auto simp add*: *final-notin-def*)
  from *termi-c' c-c' noFault-c'*
  show $\Gamma \vdash c \downarrow Normal\ s$
    by (*rule terminates-fewer-guards*)

qed

lemma *SubsetGuards*:
  assumes *c-c'*: $c \subseteq_g c'$
  assumes *deriv*: $\Gamma,\Theta\vdash_{t/\{\}}\ P\ c'\ Q,A$
  shows $\Gamma,\Theta\vdash_{t/\{\}}\ P\ c\ Q,A$
apply (*rule hoaret-complete'*)
apply (*rule SubsetGuards-sound* [*OF c-c'*])
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])
done

lemma *NormalizeD-sound*:
  assumes *valid*: $\Gamma,\Theta\models_{t/F}\ P\ (\mathit{normalize}\ c)\ Q,A$
  shows $\Gamma,\Theta\models_{t/F}\ P\ c\ Q,A$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (\mathit{Call}\ p)\ Q,A$
  assume *exec*: $\Gamma\vdash\langle c,\mathit{Normal}\ s\rangle \Rightarrow t$
  hence *exec-norm*: $\Gamma\vdash\langle \mathit{normalize}\ c,\mathit{Normal}\ s\rangle \Rightarrow t$
    by (*rule exec-to-exec-normalize*)
  assume *P*: $s \in P$
  assume *noFault*: $t \notin \mathit{Fault}\ `\ F$
  from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec-norm P noFault*]
  show $t \in \mathit{Normal}\ `\ Q \cup \mathit{Abrupt}\ `\ A$**.**
next
  fix *s*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (\mathit{Call}\ p)\ Q,A$
  assume *P*: $s \in P$
  from *cvalidt-termD* [*OF valid ctxt P*]
  have $\Gamma\vdash\mathit{normalize}\ c \downarrow \mathit{Normal}\ s$**.**
  thus $\Gamma\vdash c \downarrow \mathit{Normal}\ s$
    by (*rule terminates-normalize-to-terminates*)
qed

lemma *NormalizeD*:
  assumes *deriv*: $\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{normalize}\ c)\ Q,A$
  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 $\Gamma,\Theta\models_{t/F}\ P\ (\mathit{normalize}\ c)\ Q,A$
proof (*rule cvalidtI*)
  fix *s t*
  assume *ctxt*: $\forall (P,\ p,\ Q,\ A)\in\Theta.\ \Gamma\models_{t/F}\ P\ (\mathit{Call}\ p)\ Q,A$

assume $\Gamma\vdash\langle normalize\ c,Normal\ s\rangle \Rightarrow t$
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

13.3.5 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*)

13.3.6 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

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

lemma *DynComConseq*:
assumes $P \subseteq \{s.\ \exists P'\ Q'\ A'.\ \ \Gamma,\Theta\vdash_{t/F} P'\ (c\ s)\ Q',A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$
shows $\Gamma,\Theta\vdash_{t/F} P\ \mathit{DynCom}\ c\ Q,A$
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*:
$[\![P \subseteq \{s.\ \exists\ Z.\ s{\in}P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow\ t \in Q)\};$
$\forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ Z)\ (c\ Z)\ (Q'\ Z),\{\};$
$\forall Z.\ c\ Z = c\ \mathit{undefined}$
$]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ (\mathit{specAnno}\ P'\ c\ Q'\ (\lambda s.\ \{\}))\ Q,A$
apply (*rule SpecAnno'*)
apply *auto*
done

lemma *Skip*: $P \subseteq Q \Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ \mathit{Skip}\ Q,A$
by (*rule hoaret.Skip* [*THEN conseqPre*],*simp*)

lemma *Basic*: $P \subseteq \{s.\ (f\ s) \in Q\} \Longrightarrow\ \Gamma,\Theta\vdash_{t/F} P\ (\mathit{Basic}\ f)\ Q,A$
by (*rule hoaret.Basic* [*THEN conseqPre*])

lemma *BasicCond*:
$[\![P \subseteq \{s.\ (b\ s \longrightarrow f\ s{\in}Q) \wedge (\neg\ b\ s \longrightarrow g\ s{\in}Q)\}]\!] \Longrightarrow$
$\Gamma,\Theta\vdash_{t/F} P\ \mathit{Basic}\ (\lambda s.\ \mathit{if}\ b\ s\ \mathit{then}\ f\ s\ \mathit{else}\ g\ s)\ Q,A$
apply (*rule Basic*)
apply *auto*
done

lemma *Spec*: $P \subseteq \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t.\ (s,t) \in r)\}$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F} P\ (\mathit{Spec}\ r)\ Q,A$
by (*rule hoaret.Spec* [*THEN conseqPre*])

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

lemma *Seq* [*trans*, *intro?*]:

$[\![\Gamma,\Theta\vdash_{t/F}\ P\ c_1\ R,A;\ \Gamma,\Theta\vdash_{t/F}\ R\ c_2\ Q,A]\!] \Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ Seq\ c_1\ c_2\ Q,A$
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*:
$[\![P \subseteq \{s.\ s \in g \longrightarrow s \in R\};\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ f \in F]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{Guard}\ f\ g\ c)\ Q,A$
apply (*rule Guarantee* [*THEN conseqPre*, *of* - - - - - $\{s.\ s \in g \longrightarrow s \in R\}$])
apply *assumption*
apply (*erule conseqPre*)
apply *auto*
done

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

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

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

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

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

lemma *GuaranteeAsGuard*:
$[\![P \subseteq g \cap R;\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ \mathit{guaranteeStrip}\ f\ g\ c\ Q,A$
by (*unfold guaranteeStrip-def*) (*rule Guard*)

lemma *GuaranteeAsGuardSwap*:
$[\![\ \Gamma,\Theta\vdash_{t/F}\ R\ c\ Q,A;\ P \subseteq g \cap R]\!]$
$\Longrightarrow \Gamma,\Theta\vdash_{t/F}\ P\ \mathit{guaranteeStrip}\ f\ g\ c\ Q,A$
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*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ Z\ \wedge$
$(\forall t \in Q'\ Z.\ \mathit{return}\ s\ t \in R\ s\ t)\ \wedge$
$(\forall t \in A'\ Z.\ \mathit{return}\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ Z)\ \mathit{Call}\ p\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{call\ init\ p\ return\ c})\ Q,A$
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*: $P \subseteq \{s.\ \exists Z.\ \mathit{init}\ s \in P'\ Z\ \wedge$
$(\forall t.\ t \in Q'\ Z \longrightarrow \mathit{return}\ s\ t \in R\ s\ t)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F}\ (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall Z.\ \Gamma,\Theta\vdash_{t/F}\ (P'\ Z)\ \mathit{Call}\ p\ (Q'\ Z),\{\}$
shows $\Gamma,\Theta\vdash_{t/F}\ P\ (\mathit{call\ init\ p\ return\ c})\ Q,A$
apply (*rule ProcSpec* [*OF - c p*])
using *adapt*
apply *simp*
done

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

lemma *ProcRec*:
assumes *deriv-bodies*:
$\forall p{\in}\mathit{Procs}.$
$\forall \sigma\ Z.\ \Gamma,\Theta\cup(\bigcup q{\in}\mathit{Procs}.\ \bigcup Z.$
$\{(P\ q\ Z \cap \{s.\ ((s,q),\ \sigma,p) \in r\},q,Q\ q\ Z,A\ q\ Z)\})$
$\vdash_{t/F}\ (\{\sigma\} \cap P\ p\ Z)\ (\mathit{the}\ (\Gamma\ p))\ (Q\ p\ Z),(A\ p\ Z)$
assumes *wf*: *wf r*
assumes *Procs-defined*: $\mathit{Procs} \subseteq \mathit{dom}\ \Gamma$
shows $\forall p{\in}\mathit{Procs}.\ \forall Z.$
$\Gamma,\Theta\vdash_{t/F}(P\ p\ Z)\ \mathit{Call}\ p\ (Q\ p\ Z),(A\ p\ Z)$
by (*intro strip*)
(*rule HoareTotalDef.CallRec′*
[*OF - Procs-defined wf deriv-bodies*],
simp-all)

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

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

lemma *ProcRecSpecs*:
$[\![\forall \sigma.\ \forall (P,p,Q,A) \in Specs.$
$\Gamma,\Theta\cup\ ((\lambda(P,q,Q,A).\ (P \cap \{s.\ ((s,q),(\sigma,p)) \in r\},q,Q,A))\ `\ Specs)$
$\vdash_{t/F}(\{\sigma\} \cap P)\ (the\ (\Gamma\ p))\ Q,A;$
wf r;
$\forall (P,p,Q,A) \in Specs.\ p \in dom\ \Gamma]\!]$
$\Longrightarrow \forall (P,p,Q,A) \in Specs.\ \Gamma,\Theta\vdash_{t/F}P\ (Call\ p)\ Q,A$
apply (*rule ballI*)
apply (*case-tac x*)
apply (*rename-tac x P p Q A*)
apply *simp*
apply (*rule hoaret.CallRec*)
apply *auto*
done

lemma *ProcRec1*:
assumes *deriv-body*:
$\forall \sigma\ Z.\ \Gamma,\Theta\cup(\bigcup Z.\ \{(P\ Z \cap \{s.\ ((s,p),\ \sigma,p) \in r\},p,Q\ Z,A\ Z)\})$

$\vdash_{t/F}$ ($\{\sigma\} \cap P\ Z$) (*the* (Γ *p*)) ($Q\ Z$),($A\ Z$)
assumes *wf*: *wf r*
assumes *p-defined*: $p \in$ *dom* Γ
shows $\forall Z$. Γ,$\Theta\vdash_{t/F}$ ($P\ Z$) *Call p* ($Q\ Z$),($A\ Z$)
proof −
from *deriv-body wf p-defined*
have $\forall p{\in}\{p\}$. $\forall Z$. Γ,$\Theta\vdash_{t/F}$ ($P\ Z$) *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*:
$\forall Z$. Γ,$\Theta\vdash_{t/F}$ ($P\ Z$) (*the* (Γ *p*)) ($Q\ Z$),($A\ Z$)
assumes *p-defined*: $p \in$ *dom* Γ
shows $\forall Z$. Γ,$\Theta\vdash_{t/F}$ ($P\ Z$) *Call p* ($Q\ Z$),($A\ Z$)
proof −
have $\forall \sigma\ Z$. Γ,$\Theta\vdash_{t/F}$ ($\{\sigma\} \cap P\ Z$) (*the* (Γ *p*)) ($Q\ Z$),($A\ Z$)
by (*blast intro*: *conseqPre deriv-body* [*rule-format*])
with *p-defined* **have** $\forall \sigma\ Z$. Γ,$\Theta\cup(\bigcup Z.\ \{(P\ Z \cap \{s.\ ((s,p),\ \sigma,p) \in \{\}\},$
$p,Q\ Z,A\ Z)\})$
$\vdash_{t/F}$ ($\{\sigma\} \cap P\ Z$) (*the* (Γ *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*: Γ,$\Theta\vdash_{t/F}$ P' *body* Q,A
assumes *body*: $\Gamma\ p$ = *Some body*
shows Γ,$\Theta\vdash_{t/F}$ 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*: $P \subseteq \{s.\ init\ s \in P'\ s\}$
assumes *bdy*: $\forall s.\ \Gamma,\Theta\vdash_{t/F} (P'\ s)\ body\ \{t.\ return\ s\ t \in R\ s\ t\},\{t.\ return\ s\ t \in A\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *body*: $\Gamma\ p = Some\ body$
shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
apply (*unfold call-def*)
apply (*rule Block* [*OF adapt - c*])
apply (*rule allI*)
apply (*rule ProcBody* [**where** $\Gamma=\Gamma$, *OF - bdy* [*rule-format*] *body*])
apply *simp*
done

lemmas *ProcModifyReturn* = *HoareTotalProps.ProcModifyReturn*
lemmas *ProcModifyReturnSameFaults* = *HoareTotalProps.ProcModifyReturnSameFaults*

lemma *ProcModifyReturnNoAbr*:
  assumes *spec*: $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
  assumes *result-conform*:
    $\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
  assumes *modifies-spec*:
  $\forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV} \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
  shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturn* [*OF spec result-conform - modifies-spec*]) *simp*

lemma *ProcModifyReturnNoAbrSameFaults*:
  assumes *spec*: $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return'\ c)\ Q,A$
  assumes *result-conform*:
    $\forall s\ t.\ t \in Modif\ (init\ s) \longrightarrow (return'\ s\ t) = (return\ s\ t)$
  assumes *modifies-spec*:
  $\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ Call\ p\ (Modif\ \sigma),\{\}$
  shows $\Gamma,\Theta\vdash_{t/F} P\ (call\ init\ p\ return\ c)\ Q,A$
by (*rule ProcModifyReturnSameFaults* [*OF spec result-conform - modifies-spec*])
simp

lemma *DynProc*:
  assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ init\ s \in P'\ s\ Z\ \wedge$
    $(\forall t.\ t \in Q'\ s\ Z \longrightarrow\ return\ s\ t \in R\ s\ t)\ \wedge$
    $(\forall t.\ t \in A'\ s\ Z \longrightarrow return\ s\ t \in A)\}$
  assumes *c*: $\forall s\ t.\ \Gamma,\Theta\vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
  assumes *p*: $\forall s\in P.\ \forall Z.\ \Gamma,\Theta\vdash_{t/F} (P'\ s\ Z)\ Call\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
  shows $\Gamma,\Theta\vdash_{t/F} P\ dynCall\ init\ p\ return\ c\ Q,A$
apply (*rule conseq* [**where** $P'=\lambda Z.\ \{s.\ s=Z \wedge s \in P\}$
  and $Q'=\lambda Z.\ Q$ **and** $A'=\lambda Z.\ A$])
prefer *2*
using *adapt*

apply *blast*
apply (*rule allI*)
apply (*unfold dynCall-def call-def block-def*)
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])
apply *clarsimp*
apply (*rename-tac Z'*)
apply (*rule-tac R*=Q' Z Z' **in** *Seq*)
apply (*rule CatchSwap*)
apply (*rule SeqSwap*)
apply (*rule Throw*)
apply (*rule subset-refl*)
apply (*rule Basic*)
apply (*rule subset-refl*)
apply (*rule-tac R*=$\{i.\ i \in P'\ Z\ Z'\}$ **in** *Seq*)
apply (*rule Basic*)
apply *clarsimp*
apply *simp*
apply (*rule-tac* Q'=Q' Z Z' **and** A'=A' Z Z' **in** *conseqPost*)
using *p*
apply *clarsimp*
apply *simp*
apply *clarsimp*
apply (*rule-tac* P'=$\lambda Z''.\ \{t.\ t{=}Z'' \wedge$ *return* $Z\ t \in R\ Z\ t\}$ **and**
Q'=$\lambda Z''.\ Q$ **and** A'=$\lambda Z''.\ A$ **in** *conseq*)
prefer *2* **apply** *simp*
apply (*rule allI*)
apply (*rule HoareTotalDef.DynCom*)
apply *clarsimp*
apply (*rule SeqSwap*)
apply (*rule c* [*rule-format*])
apply (*rule Basic*)
apply *clarsimp*
done

lemma *DynProc'*:
assumes *adapt*: $P \subseteq \{s.\ \exists Z.\ \textit{init}\ s \in P'\ s\ Z\ \wedge$
$(\forall t \in Q'\ s\ Z.\ \textit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t \in A'\ s\ Z.\ \textit{return}\ s\ t \in A)\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *p*: $\forall s{\in}\ P.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ s\ Z)\ \textit{Call}\ (p\ s)\ (Q'\ s\ Z),(A'\ s\ Z)$
shows $\Gamma,\Theta \vdash_{t/F} P\ \textit{dynCall init p return c}\ Q,A$
proof −
from *adapt* **have** $P \subseteq \{s.\ \exists Z.\ \textit{init}\ s \in P'\ s\ Z\ \wedge$
$(\forall t.\ t \in Q'\ s\ Z \longrightarrow \textit{return}\ s\ t \in R\ s\ t) \wedge$
$(\forall t.\ t \in A'\ s\ Z \longrightarrow \textit{return}\ s\ t \in A)\}$

by *blast*
from *this c p* **show** *?thesis*
by (*rule DynProc*)
qed

lemma *DynProcStaticSpec*:
assumes *adapt*: $P \subseteq \{s.\ s \in S \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall s{\in}S.\ \forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ Call\ (p\ s)\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
proof −
from *adapt* **have** *P-S*: $P \subseteq S$
by *blast*
have $\Gamma,\Theta \vdash_{t/F} (P \cap S)\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProc* [**where** $P'{=}\lambda s\ Z.\ P'\ Z$ **and** $Q'{=}\lambda s\ Z.\ Q'\ Z$
and $A'{=}\lambda s\ Z.\ A'\ Z$, *OF - c*])
apply *clarsimp*
apply (*frule in-mono* [*rule-format*, *OF adapt*])
apply *clarsimp*
using *spec*
apply *clarsimp*
done
thus *?thesis*
by (*rule conseqPre*) (*insert P-S*,*blast*)
qed

lemma *DynProcProcPar*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau) \wedge$
$(\forall \tau.\ \tau \in A'\ Z \longrightarrow return\ s\ \tau \in A))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ Call\ q\ (Q'\ Z),(A'\ Z)$
shows $\Gamma,\Theta \vdash_{t/F} P\ (dynCall\ init\ p\ return\ c)\ Q,A$
apply (*rule DynProcStaticSpec* [**where** $S{=}\{s.\ p\ s = q\}$,*simplified*, *OF adapt c*])
using *spec*
apply *simp*
done

lemma *DynProcProcParNoAbrupt*:
assumes *adapt*: $P \subseteq \{s.\ p\ s = q \wedge (\exists Z.\ init\ s \in P'\ Z\ \wedge$
$(\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau))\}$
assumes *c*: $\forall s\ t.\ \Gamma,\Theta \vdash_{t/F} (R\ s\ t)\ (c\ s\ t)\ Q,A$
assumes *spec*: $\forall Z.\ \Gamma,\Theta \vdash_{t/F} (P'\ Z)\ Call\ q\ (Q'\ Z),\{\}$

shows $\Gamma,\Theta\vdash_{t/F}$ P $(dynCall\ init\ p\ return\ c)$ Q,A
proof −
  have $P \subseteq \{s.\ p\ s = q \wedge (\exists\ Z.\ init\ s \in P'\ Z \wedge$
  $(\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t) \wedge$
  $(\forall t.\ t \in \{\} \longrightarrow return\ s\ t \in A))\}$
    (**is** $P \subseteq\ ?P'$)
  proof
    fix s
    assume P: $s{\in}P$
    with *adapt* **obtain** Z **where**
      Pre: $p\ s = q \wedge init\ s \in P'\ Z$ **and**
      adapt-Norm: $\forall \tau.\ \tau \in Q'\ Z \longrightarrow return\ s\ \tau \in R\ s\ \tau$
      by *blast*
    from *adapt-Norm*
    have $\forall t.\ t \in Q'\ Z \longrightarrow return\ s\ t \in R\ s\ t$
      by *auto*
    then
    show $s{\in}?P'$
      using *Pre* **by** *blast*
  qed
  note P = *this*
  show *?thesis*
    apply −
    apply (*rule DynProcStaticSpec* [**where** S=$\{s.\ p\ s = q\}$,*simplified*, *OF P c*])
    apply (*insert spec*)
    apply *auto*
    done
qed

lemma *DynProcModifyReturnNoAbr*:
  assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}$ P $(dynCall\ init\ p\ return'\ c)$ Q,A
  assumes *ret-nrm-modif*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
  assumes *modif-clause*:
  $\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/UNIV}$ $\{\sigma\}$ $Call\ (p\ s)$ $(Modif\ \sigma)$,$\{\}$
  shows $\Gamma,\Theta\vdash_{t/F}$ P $(dynCall\ init\ p\ return\ c)$ Q,A
proof −
  from *ret-nrm-modif*
  have $\forall s\ t.\ t\ \in (Modif\ (init\ s))$
    $\longrightarrow return'\ s\ t = return\ s\ t$
    by *iprover*
  then
  have *ret-nrm-modif'*: $\forall s\ t.\ t \in (Modif\ (init\ s))$
  $\longrightarrow return'\ s\ t = return\ s\ t$
    by *simp*
  have *ret-abr-modif'*: $\forall s\ t.\ t \in \{\}$
  $\longrightarrow return'\ s\ t = return\ s\ t$
    by *simp*
  from *to-prove ret-nrm-modif' ret-abr-modif' modif-clause* **show** *?thesis*

by (*rule dynProcModifyReturn*)
qed

lemma *ProcDynModifyReturnNoAbrSameFaults*:
assumes *to-prove*: $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return′ c*) Q,A
assumes *ret-nrm-modif*: $\forall s\ t.\ t \in$ (*Modif* (*init s*))
$\longrightarrow$ *return′ s t* = *return s t*
assumes *modif-clause*:
$\forall s \in P.\ \forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}$ (*Call* (*p s*)) (*Modif* σ),{}
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* σ),(*ModifAbr* σ)
shows $\Gamma,\Theta\vdash_{t/F}$ P (*dynCall init p return c*) Q,A
proof −
from *to-prove* **have** $\Gamma,\Theta\vdash_{t/F}$ $(\{s.\ p\ s = q\} \cap P')$ (*dynCall init p return′ c*) Q,A
by (*rule conseqPre*) *blast*
from *this ret-nrm-modif*
ret-abr-modif
have $\Gamma,\Theta\vdash_{t/F}$ $(\{s.\ p\ s = q\} \cap P')$ (*dynCall init p return c*) Q,A
by (*rule 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*:
      $\forall \sigma.\ \Gamma,\Theta\vdash_{/F} \{\sigma\}\ (Call\ q)\ (Modif\ \sigma),\{\}$
  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 ProcDynModifyReturnNoAbrSameFaults*) (*insert modif-clause*,*auto*)
  from *this q* **show** *?thesis*
    by (*rule conseqPre*)
qed

lemma *MergeGuards-iff*: $\Gamma,\Theta\vdash_{t/F} P\ merge\text{-}guards\ c\ Q,A = \Gamma,\Theta\vdash_{t/F} P\ c\ Q,A$
  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'': $c''= mark\text{-}guards\ False\ (strip\text{-}guards\ (-F)\ c')$
  assumes *c*: $merge\text{-}guards\ c = merge\text{-}guards\ (mark\text{-}guards\ False\ c')$
  shows $\Gamma,\Theta\vdash_{t/\{\}} P\ c\ Q,A$
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 $\Gamma,\Theta\vdash_{/\{\}} P\ (strip\text{-}guards\ (-\ F)\ c')\ UNIV,UNIV$
    by (*rule HoarePartialProps.MarkGuardsD*)
  with *deriv*
  have $\Gamma,\Theta\vdash_{t/\{\}} P\ c'\ Q,A$
    by (*rule CombineStrip*)
  hence $\Gamma,\Theta\vdash_{t/\{\}} P\ mark\text{-}guards\ False\ c'\ Q,A$
    by (*rule MarkGuardsI*)
  hence $\Gamma,\Theta\vdash_{t/\{\}} P\ merge\text{-}guards\ (mark\text{-}guards\ False\ c')\ Q,A$
    by (*rule MergeGuardsI*)
  hence $\Gamma,\Theta\vdash_{t/\{\}} P\ merge\text{-}guards\ c\ Q,A$
    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

14.0.1 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

15 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*-,-/$\vdash_{t'/\text{-}}$ (-/ (-)/ -)) [*61*,*60*,*60*,*1000*,*20*,*1000*]*60*)

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

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

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

syntax (*ASCII*)

-hoarep-emptyFaults::
[(*'s,'p,'f*) *body*,(*'s,'p*) *quadruple set*,
's assn,(*'s,'p,'f*) *com*, *'s assn*,*'s assn*] ⇒ *bool*
((*3*-,-/|− (-/ (-)/ -,/-)) [*61*,*60*,*1000*,*20*,*1000*,*1000*]*60*)

-hoarep-emptyCtx::
[(*'s,'p,'f*) *body*,*'f set*,*'s assn*,(*'s,'p,'f*) *com*, *'s assn*,*'s assn*] => *bool*
((*3*-/|−'/- (-/ (-)/ -,/-)) [*61*,*60*,*1000*,*20*,*1000*,*1000*]*60*)

-hoarep-emptyCtx-emptyFaults::
[(*'s,'p,'f*) *body*,*'s assn*,(*'s,'p,'f*) *com*, *'s assn*,*'s assn*] => *bool*
((*3*-/|−(-/ (-)/ -,/-)) [*61*,*1000*,*20*,*1000*,*1000*]*60*)

-hoarep-noAbr::
[(*'s,'p,'f*) *body*,(*'s,'p*) *quadruple set*,*'f set*,
's assn,(*'s,'p,'f*) *com*, *'s assn*] => *bool*
((*3*-,-/|−'/- (-/ (-)/ -)) [*61*,*60*,*60*,*1000*,*20*,*1000*]*60*)

-hoarep-noAbr-emptyFaults::
[(*'s,'p,'f*) *body*,(*'s,'p*) *quadruple set*,*'s assn*,(*'s,'p,'f*) *com*, *'s assn*] => *bool*
((*3*-,-/|−(-/ (-)/ -)) [*61*,*60*,*1000*,*20*,*1000*]*60*)

-hoarep-emptyCtx-noAbr::
[(*'s,'p,'f*) *body*,*'f set*,*'s assn*,(*'s,'p,'f*) *com*, *'s assn*] => *bool*
((*3*-/|−'/- (-/ (-)/ -)) [*61*,*60*,*1000*,*20*,*1000*]*60*)

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

-hoaret-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

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

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

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

$\Gamma\vdash\ P\ c\ Q \quad <= \ \Gamma\vdash\ P\ c\ Q,\{\}$

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

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

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

term $\Gamma\vdash\ P\ c\ Q$
term $\Gamma\vdash\ P\ c\ Q,A$

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

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

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

term $\Gamma\vdash_t\ P\ c\ Q$
term $\Gamma\vdash_t\ P\ c\ Q,A$

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

term $\Gamma,\Theta\vdash\ P\ c\ Q$
term $\Gamma,\Theta\vdash_{t/F}\ P\ c\ Q$

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

locale *hoare* =
 fixes Γ::$('s,'p,'f)$ *body*

primrec *assoc*:: $('a \times 'b)$ *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*: $(x,y) \in \mathit{inv\text{-}image}\ r\ f = ((f\,x,\ f\,y) \in r)$
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 $\mathit{wf}\ (f <*mlex*> r)$
proof (*rule ccontr*)
assume $\neg\ \mathit{wf}\ (f <*mlex*> r)$
then
obtain *g* **where** $\forall i.\ (g\ (\mathit{Suc}\ i),\ g\ i) \in f <*mlex*> r$
by (*auto simp add*: *wf-iff-no-infinite-down-chain*)
hence *g*: $\forall i.\ (g\ (\mathit{Suc}\ i),\ g\ i) \in \mathit{measure}\ f\ \vee$
$f\ (g\ (\mathit{Suc}\ i)) = f\ (g\ i) \wedge (g\ (\mathit{Suc}\ i),\ g\ i) \in r$
by (*simp add*: *measure-lex-prod-def′*)
hence *le-g*: $\forall i.\ f\ (g\ (\mathit{Suc}\ i)) \le f\ (g\ i)$
by (*auto simp add*: *in-measure-iff order-le-less*)
have *wf* (*measure f*)
by *simp*
hence $\forall Q.\ (\exists x.\ x \in Q) \longrightarrow (\exists z{\in}Q.\ \forall y.\ (y,\ z) \in \mathit{measure}\ f \longrightarrow y \notin Q)$
by (*simp add*: *wf-eq-minimal*)
from *this* [*rule-format*, *of g ‘ UNIV*]
have $\exists z.\ z \in \mathit{range}\ g \wedge (\forall y.\ (y,\ z) \in \mathit{measure}\ f \longrightarrow y \notin \mathit{range}\ g)$
by *auto*
then obtain *z* **where**
z: $z \in \mathit{range}\ g$ **and**
min-z: $\forall y.\ f\,y < f\,z \longrightarrow y \notin \mathit{range}\ g$
by (*auto simp add*: *in-measure-iff*)
from *z* **obtain** *k* **where**
k: $z = g\ k$
by *auto*
have $\forall i.\ k \le i \longrightarrow f\ (g\ i) = f\ (g\ k)$
proof (*intro allI impI*)
fix *i*
assume $k \le i$ **then show** $f\ (g\ i) = f\ (g\ k)$
proof (*induct i*)
case *0*
have $k \le 0$ **by** *fact* **hence** $k = 0$ **by** *simp*
thus $f\ (g\ 0) = f\ (g\ k)$
by *simp*
next
case (*Suc n*)
have *k-Suc-n*: $k \le \mathit{Suc}\ n$ **by** *fact*
then show $f\ (g\ (\mathit{Suc}\ n)) = f\ (g\ k)$
proof (*cases* $k = \mathit{Suc}\ n$)

```
        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

16 State Space Template

theory *StateSpace* **imports** *Hoare*
begin

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

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

record (*'g*, *'n*, *'val*) *stateSP* = *'g state* +
locals :: *'n* ⇒ *'val*

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

end

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

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

lemma *protectImp*:
assumes *i*: *PROP Pure.prop* (*PROP P* ⟹ *PROP Q*)
shows *PROP Pure.prop* (*PROP Pure.prop P* ⟹ *PROP Pure.prop Q*)
proof −
{
assume *P*: *PROP Pure.prop P*
from *i* [*unfolded prop-def*, *OF P* [*unfolded prop-def*]]
have *PROP Pure.prop Q*
by (*simp add*: *prop-def*)
}
note *i'* = *this*
show *PROP ?thesis*
apply (*rule protectI*)
apply (*rule i'*)
apply *assumption*

done
qed

lemma *generaliseConj*:
assumes *i1*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop Q*))
assumes *i2*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P'*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop Q'*))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($P \wedge P'$)) $\Longrightarrow$ (*PROP Pure.prop* (*Trueprop* ($Q \wedge Q'$))))
using *i1 i2*
by (*auto simp add*: *prop-def*)

lemma *generaliseAll*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s$. *PROP Pure.prop* (*Trueprop* (*P s*)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* (*Q s*)))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($\forall s.\ P\ s$)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\forall s.\ Q\ s$)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generalise-all*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s$. *PROP Pure.prop* (*PROP P s*) $\Longrightarrow$ *PROP Pure.prop* (*PROP Q s*))
shows *PROP Pure.prop* ((*PROP Pure.prop* ($\bigwedge s$. *PROP P s*)) $\Longrightarrow$ (*PROP Pure.prop* ($\bigwedge s$. *PROP Q s*)))
using *i*
proof (*unfold prop-def*)
assume *i1*: $\bigwedge s$. (*PROP P s*) $\Longrightarrow$ (*PROP Q s*)
assume *i2*: $\bigwedge s$. *PROP P s*
show $\bigwedge s$. *PROP Q s*
by (*rule i1*) (*rule i2*)
qed

lemma *generaliseTrans*:
assumes *i1*: *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP Q*)
assumes *i2*: *PROP Pure.prop* (*PROP Q* $\Longrightarrow$ *PROP R*)
shows *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP R*)
using *i1 i2*
proof (*unfold prop-def*)
assume *P-Q*: *PROP P* $\Longrightarrow$ *PROP Q*
assume *Q-R*: *PROP Q* $\Longrightarrow$ *PROP R*
assume *P*: *PROP P*
show *PROP R*
by (*rule Q-R* [*OF P-Q* [*OF P*]])
qed

lemma *meta-spec*:

assumes $\bigwedge x.$ *PROP P x*
shows *PROP P x* **by** *fact*

lemma *meta-spec-protect*:
assumes *g*: $\bigwedge x.$ *PROP P x*
shows *PROP Pure.prop* (*PROP P x*)
using *g*
by (*auto simp add*: *prop-def*)

lemma *generaliseImp*:
assumes *i*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop Q*))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($X \longrightarrow P$)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($X \longrightarrow Q$)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generaliseEx*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s.$ *PROP Pure.prop* (*Trueprop* (*P s*)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* (*Q s*)))
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop* ($\exists s.\ P\ s$)) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\exists s.\ Q\ s$)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generaliseRefl*: *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop P*))
by (*auto simp add*: *prop-def*)

lemma *generaliseRefl'*: *PROP Pure.prop* (*PROP P* $\Longrightarrow$ *PROP P*)
by (*auto simp add*: *prop-def*)

lemma *generaliseAllShift*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s.\ P \Longrightarrow Q\ s$)
shows *PROP Pure.prop* (*PROP Pure.prop* (*Trueprop P*) $\Longrightarrow$ *PROP Pure.prop* (*Trueprop* ($\forall s.\ Q\ s$)))
using *i*
by (*auto simp add*: *prop-def*)

lemma *generalise-allShift*:
assumes *i*: *PROP Pure.prop* ($\bigwedge s.$ *PROP P* $\Longrightarrow$ *PROP Q s*)
shows *PROP Pure.prop* (*PROP Pure.prop* (*PROP P*) $\Longrightarrow$ *PROP Pure.prop* ($\bigwedge s.$ *PROP Q s*))
using *i*
proof (*unfold prop-def*)
assume *P-Q*: $\bigwedge s.$ *PROP P* $\Longrightarrow$ *PROP Q s*
assume *P*: *PROP P*
show $\bigwedge s.$ *PROP Q s*

by (*rule P-Q* [*OF P*])
qed

lemma *generaliseImpl*:
assumes *i*: *PROP Pure.prop* (*PROP Pure.prop P* $\Longrightarrow$ *PROP Pure.prop Q*)
shows *PROP Pure.prop* ((*PROP Pure.prop* (*PROP X* $\Longrightarrow$ *PROP P*)) $\Longrightarrow$ (*PROP Pure.prop* (*PROP X* $\Longrightarrow$ *PROP Q*)))
using *i*
proof (*unfold prop-def*)
assume *i1*: *PROP P* $\Longrightarrow$ *PROP Q*
assume *i2*: *PROP X* $\Longrightarrow$ *PROP P*
assume *X*: *PROP X*
show *PROP Q*
by (*rule i1* [*OF i2* [*OF X*]])
qed

ML-file *generalise-state.ML*

end

17 Facilitating the Hoare Logic

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

17.1 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*] $\Rightarrow ('a,'b,'f)$ *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* $\Rightarrow ('a \times 'a)$ *set* $\Rightarrow$ *bdy*
$\Rightarrow ('a,'p,'f)$ *com*
((*0WHILE* (-)/ *INV* (-)/ *VAR* (-) /-) [*25*, *0*, *0*, *81*] *71*)
-*WhileFix-inv-var* :: $'a$ *bexp* => *pttrn* $\Rightarrow ('z \Rightarrow 'a$ *assn*$)$ $\Rightarrow$
$('z \Rightarrow ('a \times 'a)$ *set*$)$ $\Rightarrow$ *bdy*
$\Rightarrow ('a,'p,'f)$ *com*
((*0WHILE* (-)/ *FIX* -./ *INV* (-)/ *VAR* (-) /-) [*25*, *0*, *0*, *0*, *81*] *71*)
-*WhileFix-inv* :: $'a$ *bexp* => *pttrn* $\Rightarrow ('z \Rightarrow 'a$ *assn*$)$ $\Rightarrow$ *bdy*

$\Rightarrow$ ('a,'p,'f) com
((0WHILE (-)/ FIX -./ INV (-) /-) [25, 0, 0, 81] 71)
-GuardedWhileFix-inv-var :: 'a bexp => pttrn $\Rightarrow$ ('z $\Rightarrow$ 'a assn) $\Rightarrow$
('z $\Rightarrow$ ('a $\times$ 'a) set) $\Rightarrow$ bdy
$\Rightarrow$ ('a,'p,'f) com
((0WHILE$_g$ (-)/ FIX -./ INV (-)/ VAR (-) /-) [25, 0, 0, 0, 81] 71)
-GuardedWhileFix-inv-var-hook :: 'a bexp $\Rightarrow$ ('z $\Rightarrow$ 'a assn) $\Rightarrow$
('z $\Rightarrow$ ('a $\times$ 'a) set) $\Rightarrow$ bdy
$\Rightarrow$ ('a,'p,'f) com
-GuardedWhileFix-inv :: 'a bexp => pttrn $\Rightarrow$ ('z $\Rightarrow$ 'a assn) $\Rightarrow$ bdy
$\Rightarrow$ ('a,'p,'f) com
((0WHILE$_g$ (-)/ FIX -./ INV (-)/-) [25, 0, 0, 81] 71)

-GuardedWhile-inv-var::
'a bexp => 'a assn $\Rightarrow$ ('a $\times$ 'a) set $\Rightarrow$ bdy $\Rightarrow$ ('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 (-/$\longmapsto$ (1-)) /-) [1000,25,81] 71)
-While-guard-inv:: grds $\Rightarrow$'a bexp$\Rightarrow$'a assn$\Rightarrow$bdy $\Rightarrow$ ('a,'p,'f) com
((0WHILE (-/$\longmapsto$ (1-)) INV (-) /-) [1000,25,0,81] 71)
-While-guard-inv-var:: grds $\Rightarrow$'a bexp$\Rightarrow$'a assn$\Rightarrow$('a$\times$'a) set
$\Rightarrow$bdy $\Rightarrow$ ('a,'p,'f) com
((0WHILE (-/$\longmapsto$ (1-)) INV (-)/ VAR (-) /-) [1000,25,0,0,81] 71)
-WhileFix-guard-inv-var:: grds $\Rightarrow$'a bexp$\Rightarrow$pttrn$\Rightarrow$('z$\Rightarrow$'a assn)$\Rightarrow$('z$\Rightarrow$('a$\times$'a) set)
$\Rightarrow$bdy $\Rightarrow$ ('a,'p,'f) com
((0WHILE (-/$\longmapsto$ (1-)) FIX -./ INV (-)/ VAR (-) /-) [1000,25,0,0,0,81] 71)
-WhileFix-guard-inv:: grds $\Rightarrow$'a bexp$\Rightarrow$pttrn$\Rightarrow$('z$\Rightarrow$'a assn)
$\Rightarrow$bdy $\Rightarrow$ ('a,'p,'f) com
((0WHILE (-/$\longmapsto$ (1-)) FIX -./ INV (-)/-) [1000,25,0,0,81] 71)

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

-DoPre :: ('a,'p,'f) com $\Rightarrow$ ('a,'p,'f) com
-Do :: ('a,'p,'f) com $\Rightarrow$ bdy ((2DO/ (-)) /OD [0] 1000)
-Lab:: 'a bexp $\Rightarrow$ ('a,'p,'f) com $\Rightarrow$ bdy
(-$\cdot$/- [1000,71] 81)
:: bdy $\Rightarrow$ ('a,'p,'f) com (-)
-Spec:: pttrn $\Rightarrow$'s set $\Rightarrow$ ('s,'p,'f) com $\Rightarrow$'s set $\Rightarrow$'s set $\Rightarrow$ ('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(op = 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*) ($\lambda i.$ *c*)
-FCall p acts i c == *-FCall p acts* ($\lambda i.$ *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
theory *TimSortProc*
 imports *../Simpl/Vcg Main ~~/src/HOL/Library/Code-Target-Numeral TimSortLemma*
begin

hoarestate *globals-var =*
```
 stack-size:: nat
 run-base :: nat list
 run-len :: nat list
 stack-len :: nat
 a :: int list
 global-min-gallop :: nat
```

procedures (**imports** *globals-var*)
gallop-left(key::int,array::int list,base::nat,len::nat,hint::nat|ret::nat)
where *last-ofs::nat ofs::nat max-ofs::nat tmp-gallop::nat mid::nat* **in**

```
´last-ofs:==0;;
´ofs:==1;;
IF ´key>´array!(´base+´hint)
THEN
 ´max-ofs :== ´len − ´hint;;
 WHILE (´ofs < ´max-ofs & ´key > ´array!(´base+´hint+´ofs))
 DO
  ´last-ofs :== ´ofs;;
  ´ofs :== ´ofs+´ofs+1
 OD ;;
 IF ´ofs > ´max-ofs THEN ´ofs :== ´max-ofs FI ;;
 ´last-ofs :== ´last-ofs + ´hint+1;;
 ´ofs :== ´ofs + ´hint
ELSE
```

```
 ´max-ofs :== ´hint + 1;;
 WHILE (´ofs < ´max-ofs & ´key ≤ ´array!(´base+´hint−´ofs))
 DO
  ´last-ofs :== ´ofs;;
  ´ofs :== ´ofs+´ofs+1
 OD ;;
 IF ´ofs > ´max-ofs THEN ´ofs :== ´max-ofs FI ;;
 ´tmp-gallop :== ´last-ofs;;
 ´last-ofs :== ´hint+1 − ´ofs;;
 ´ofs :== ´hint − ´tmp-gallop
FI ;;
WHILE (´last-ofs < ´ofs)
DO
 ´mid :==  (´ofs + ´last-ofs)div 2;;
 IF (´key > ´array!(´base+´mid))
 THEN
  ´last-ofs :== ´mid+1
 ELSE
  ´ofs :== ´mid
 FI
OD;;
´ret :== ´ofs
```

procedures (**imports** *globals-var*)
gallop-right(*key*::*int*,*array*::*int list*,*base*::*nat*,*len*::*nat*,*hint*::*nat*|*ret*::*nat*)
where *last-ofs*::*nat ofs*::*nat max-ofs*::*nat tmp-gallop*::*nat mid*::*nat* **in**

```
(∗ ´stack-len :== ´stack-len;;  unnecessary but the spec need globals be modified ∗)
´last-ofs:==0;;
´ofs:==1;;
IF ´key<´array!(´base+´hint)
THEN
 ´max-ofs :== ´hint + 1;;
 WHILE (´ofs < ´max-ofs & ´key < ´array!(´base+´hint−´ofs))
 DO
  ´last-ofs :== ´ofs;;
  ´ofs :== ´ofs+´ofs+1
 OD ;;
 IF ´ofs > ´max-ofs THEN ´ofs :== ´max-ofs FI ;;
 ´tmp-gallop :== ´last-ofs;;
 ´last-ofs :== ´hint+1 − ´ofs;;
 ´ofs :== ´hint − ´tmp-gallop
ELSE
 ´max-ofs :== ´len − ´hint;;
 WHILE (´ofs < ´max-ofs & ´key ≥ ´array!(´base+´hint+´ofs))
 DO
  ´last-ofs :== ´ofs;;
  ´ofs :== ´ofs+´ofs+1
```

```
 OD ;;
 IF ´ofs > ´max-ofs THEN ´ofs :== ´max-ofs FI ;;
 ´last-ofs :== ´last-ofs + ´hint+1;;
 ´ofs :== ´ofs + ´hint
FI ;;
WHILE (´last-ofs < ´ofs)
DO
 ´mid :== (´ofs + ´last-ofs)div 2;;
 IF (´key < ´array!(´base+´mid))
 THEN
  ´ofs :== ´mid
 ELSE
  ´last-ofs :== ´mid+1
 FI
OD;;
´ret :== ´ofs
```

```
value replicate (3::nat) (4::nat)
value list-copy [1,2,3::int] 6 [1,2,3::int] 1 2
procedures (imports globals-var)
merge-lo (base1::nat, len1::nat, base2::nat, len2::nat)
where tmp::int list cursor1::nat cursor2::nat dest::nat
min-gallop::nat count1::nat count2::nat in

TRY
  ´tmp :== replicate ´len1 (0::int);;
  ´tmp :== list-copy ´tmp (0::nat) ´a ´base1 ´len1;;
  ´cursor1 :== 0;;
  ´cursor2 :==´base2;;
  ´dest :== ´base1;;
  ´a!´dest :== ´a!´cursor2;;
  ´dest :== ´dest+1;;
  ´cursor2 :== ´cursor2+1;;
  ´len2 :== ´len2−1;;
  ´min-gallop :== ´global-min-gallop;;(∗ need to add min gallop to globals var ∗)
  IF ´len2=0 THEN ´a :== list-copy ´a ´dest ´tmp ´cursor1 ´len1;; THROW
FI;;
 IF ´len1=1 THEN ´a :== list-copy ´a ´dest ´a ´cursor2 ´len2;; ´a!(´dest+´len2):==
´tmp!´cursor1;;THROW FI;;
  TRY
    WHILE True
    DO
      ´count1 :== 0;;
      ´count2 :== 0;;
      WHILE (´count1 < ´min-gallop & ´count2 < ´min-gallop)
      DO
        IF ´a!´cursor2 < ´tmp!´cursor1
```

```
  THEN
    ´a!´dest :== ´a!´cursor2;;
    ´dest :== ´dest+1;;
    ´cursor2 :== ´cursor2+1;;
    ´count2 :== ´count2+1;;
    ´count1 :== 0;;
    ´len2 :== ´len2−1;;
    IF ´len2 = 0 THEN THROW FI
  ELSE
    ´a!´dest :== ´tmp!´cursor1;;
    ´dest :== ´dest+1;;
    ´cursor1 :== ´cursor1+1;;
    ´count1 :== ´count1+1;;
    ´count2 :== 0;;
    ´len1 :== ´len1−1;;
    IF ´len1 = 1 THEN THROW FI
  FI
OD;;
WHILE (´count1 ≥ ´global-min-gallop | ´count2 ≥ ´global-min-gallop)
DO
  ´count1 :== CALL gallop-right(´a!´cursor2, ´tmp, ´cursor1, ´len1, 0);;
  IF ´count1 ≠ 0
  THEN
    ´a :== list-copy ´a ´dest ´tmp ´cursor1 ´count1;;
    ´dest :== ´dest+´count1;;
    ´cursor1 :== ´cursor1+´count1;;
    ´len1 :== ´len1−´count1;;
    IF ´len1≤1 THEN THROW FI
  FI;;
  ´a!´dest :== ´a!´cursor2;;
  ´dest :== ´dest+1;;
  ´cursor2 :== ´cursor2+1;;
  ´len2 :== ´len2−1;;
  IF ´len2 = 0 THEN THROW FI;;

  ´count2 :== CALL gallop-left(´tmp!´cursor1, ´a, ´cursor2, ´len2, 0);;
  IF ´count2 ≠ 0
  THEN
    ´a :== list-copy ´a ´dest ´a ´cursor2 ´count2;;
    ´dest :== ´dest+´count2;;
    ´cursor2 :== ´cursor2+´count2;;
    ´len2 :== ´len2−´count2;;
    IF ´len2 = 0 THEN THROW FI
  FI;;
  ´a!´dest :== ´tmp!´cursor1;;
  ´dest :== ´dest+1;;
  ´cursor1 :== ´cursor1+1;;
  ´len1 :== ´len1−1;;
  IF ´len1 = 1 THEN THROW FI;;
```

```
      ´min-gallop :== ´min-gallop−1
    OD;;
    IF ´min-gallop < 0 THEN ´min-gallop :== 0 FI;;
    ´min-gallop :== ´min-gallop + 2
  OD
 CATCH
 Skip
 END;;
 IF ´len1 = 1
  THEN ´a :== list-copy ´a ´dest ´a ´cursor2 ´len2;;´a!(´dest+´len2) :==
´tmp!´cursor1
 ELSE IF ´len1 = 0
     THEN Skip
     ELSE ´a :== list-copy ´a ´dest ´tmp ´cursor1 ´len1
     FI
 FI
CATCH
Skip
END
```

procedures (**imports** *globals-var*)
merge-hi (*base1*::*nat*, *len1*::*nat*, *base2*::*nat*, *len2*::*nat*)
where *tmp*::*int list cursor1*::*nat cursor2*::*nat dest*::*nat*
min-gallop::*nat count1*::*nat count2*::*nat count-tmp*::*nat* **in**

```
TRY
  ´tmp :== replicate ´len2 (0::int);;
  ´tmp :== list-copy ´tmp (0::nat) ´a ´base2 ´len2;;
  ´cursor1 :== ´base1+´len1−1;;
  ´cursor2 :==´len2−1;;
  ´dest :== ´base2+´len2−1;;
  ´a!´dest :== ´a!´cursor1;;
  ´dest :== ´dest−1;;
  ´cursor1 :== ´cursor1−1;;
  ´len1 :== ´len1−1;;
  ´min-gallop :== ´global-min-gallop;;(∗ need to add min gallop to globals var ∗)
  IF ´len1=0 THEN ´a :== list-copy ´a (´dest−(´len2−1)) ´tmp 0 ´len2;;
THROW FI;;
 IF ´len2=1
 THEN
   ´dest :== ´dest−´len1;;
   ´cursor1 :== ´cursor1−´len1;;
   ´a :== list-copy ´a (´dest+1) ´a (´cursor1+1) ´len1;;
   ´a!´dest:== ´tmp!´cursor2;;
   THROW
 FI;;
 TRY
   WHILE True
   DO
```

```
´count1 :== 0;;
´count2 :== 0;;
WHILE (´count1 < ´min-gallop & ´count2 < ´min-gallop)
DO
  IF ´tmp!´cursor2 < ´a!´cursor1
  THEN
    ´a!´dest :== ´a!´cursor1;;
    ´dest :== ´dest−1;;
    ´cursor1 :== ´cursor1−1;;
    ´count1 :== ´count1+1;;
    ´count2 :== 0;;
    ´len1 :== ´len1−1;;
    IF ´len1 = 0 THEN THROW FI
  ELSE
    ´a!´dest :== ´tmp!´cursor2;;
    ´dest :== ´dest−1;;
    ´cursor2 :== ´cursor2−1;;
    ´count2 :== ´count2+1;;
    ´count1 :== 0;;
    ´len2 :== ´len2−1;;
    IF ´len2 = 1 THEN THROW FI
  FI
OD;;
WHILE (´count1 ≥ ´global-min-gallop | ´count2 ≥ ´global-min-gallop)
DO
    ´count-tmp :==  CALL gallop-right(´tmp!´cursor2, ´a, ´base1, ´len1,
´len1−1);;
  ´count1 :== ´len1 − ´count-tmp;;
  IF ´count1 ≠ 0
  THEN
    ´dest :== ´dest−´count1;;
    ´cursor1 :== ´cursor1−´count1;;
    ´len1 :== ´len1−´count1;;
    ´a :== list-copy ´a (´dest+1) ´a (´cursor1+1) ´count1;;
    IF ´len1=0 THEN THROW FI
  FI;;
  ´a!´dest :== ´tmp!´cursor2;;
  ´dest :== ´dest−1;;
  ´cursor2 :== ´cursor2−1;;
  ´len2 :== ´len2−1;;
  IF ´len2 = 1 THEN THROW FI;;

 ´count-tmp :== CALL gallop-left(´a!´cursor1, ´tmp, 0, ´len2, (´len2−1));;
  ´count2 :== ´len2 − ´count-tmp;;
  IF ´count2 ≠ 0
  THEN
    ´dest :== ´dest−´count2;;
    ´cursor2 :== ´cursor2−´count2;;
    ´len2 :== ´len2−´count2;;
```

```
          ´a :== list-copy ´a (´dest+1) ´tmp (´cursor2+1) ´count2;;
          IF ´len2 ≤ 1 THEN THROW FI
        FI;;
        ´a!´dest :== ´a!´cursor1;;
        ´dest :== ´dest−1;;
        ´cursor1 :== ´cursor1−1;;
        ´len1 :== ´len1−1;;
        IF ´len1 = 0 THEN THROW FI;;
        ´min-gallop :== ´min-gallop−1
      OD;;
      IF ´min-gallop < 0 THEN ´min-gallop :== 0 FI;;
      ´min-gallop :== ´min-gallop + 2
    OD
  CATCH
  Skip
  END;;
  IF ´len2 = 1
  THEN
    ´dest :== ´dest−´len1;;
    ´cursor1 :== ´cursor1−´len1;;
    ´a :== list-copy ´a (´dest+1) ´a (´cursor1+1) ´len1;;
    ´a!´dest :== ´tmp!´cursor2
  ELSE IF ´len2 = 0
       THEN Skip
       ELSE ´a :== list-copy ´a (´dest−(´len2−1)) ´tmp 0 ´len2
       FI
  FI
CATCH
Skip
END
```

procedures (**imports** *globals-var*)
merge-at (*i*::*nat*)
where *k*::*nat base1*::*nat base2*::*nat len1*::*nat len2*::*nat* **in**

```
TRY
  ´base1 :== ´run-base!´i;;
  ´len1 :== ´run-len!´i;;
  ´base2 :== ´run-base!(´i+1);;
  ´len2 :== ´run-len!(´i+1);;
  (´run-len!´i) :== ´len1 + ´len2;;
  IF ´i=´stack-size−3
  THEN ´run-base!(´i+1) :== ´run-base!(´i+2);;
       ´run-len!(´i+1) :==(´run-len!(´i+2)) FI;;
  ´stack-size :== ´stack-size−1 ;;
  (∗ the process of merge on the array ∗)
  ´k :== CALL gallop-right(´a!´base2, ´a, ´base1, ´len1, 0);;
  ´base1 :== ´base1+´k;;
```

```
  ´len1 :== ´len1−´k;;
  IF ´len1=0 THEN THROW FI;;
  ´len2 :== CALL gallop-left(´a!(´base1+´len1−1),´a, ´base2, ´len2, ´len2−1);;
  IF ´len2=0 THEN THROW FI;;
  IF (´len1≤´len2)
  THEN CALL merge-lo(´base1, ´len1, ´base2, ´len2)
  ELSE CALL merge-hi(´base1, ´len1, ´base2, ´len2)
  FI
CATCH Skip END
```

print-locale *merge-at-impl*

procedures (**imports** *globals-var*)
merge-collapse()
where *n*::*nat* **in**

```
TRY
  WHILE ´stack-size > 1
  DO
    ´n :== ´stack-size−2;;
    IF (´n>0 ∧ ´run-len!(´n−1) ≤ ´run-len!´n + ´run-len!(´n+1))
    ∨ (´n>1 ∧ ´run-len!(´n−2) ≤ ´run-len!(´n−1) + ´run-len!´n)
    THEN
      IF ´run-len!(´n−1) < ´run-len!(´n+1)
      THEN
        ´n:==´n−1
      FI
    ELSE
      IF ´n<0 ∨ ´run-len!´n > ´run-len!(´n+1)
      THEN
        THROW
      FI
    FI;;
    CALL merge-at(´n)
  OD
CATCH SKIP END
```

print-locale *merge-collapse-impl*

procedures (**imports** *globals-var*)
push-run(*run-base-i*::*nat*, *run-len-i*::*nat*)

```
´run-base!´stack-size :== ´run-base-i ;;
 ´run-len!´stack-size :== ´run-len-i ;;
 ´stack-size :== ´stack-size + 1
```

procedures (**imports** *globals-var*)
merge-force-collapse()
where *n*::*nat* **in**

```
WHILE ´stack-size > 1
DO
```

```
  ´n :== ´stack-size − 2;;
  IF (´n > 0 ∧ ´run-len!(´n−1) < ´run-len!(´n+1))
  THEN
    ´n:==´n − 1
  FI;;
  CALL merge-at(´n)
OD
```

procedures (**imports** *globals-var*)
reverse-range(*array*::*int list*, *lo*::*nat*, *hi*::*nat*| *ret*::*int list*)
where *t*::*int* **in**

```
TRY
IF ´hi = 0 THEN THROW ELSE (∗ to address the problem of natural number ∗)
´hi:==´hi−1;;
WHILE ´lo < ´hi DO
 ´t:==´array!´lo;;
 ´array!´lo :== ´array!´hi;;
 ´array!´hi :== ´t;;
 ´lo :== ´lo+1;;
 ´hi :== ´hi−1
OD FI
CATCH Skip END;;
´ret :== ´array
```

procedures (**imports** *globals-var*)
count-run-and-make-ascending(*array*::*int list*, *lo*::*nat*, *hi*::*nat*| *ret-value*::*nat*, *ret*::*int list*)
where *run-hi*::*nat* **in**

```
´run-hi:==´lo+1;;
TRY
 IF ´run-hi = ´hi THEN ´ret-value:==1;;´ret :== ´array;;THROW FI;;
 IF ´array!´run-hi < ´array!´lo
 THEN
  ´run-hi :== ´run-hi+1;;
  WHILE (´run-hi < ´hi & ´array!´run-hi < ´array!(´run-hi−1))
  DO ´run-hi :== ´run-hi+1 OD;;
  ´array :== CALL reverse-range(´array, ´lo, ´run-hi)
 ELSE
  ´run-hi :== ´run-hi+1;;
  WHILE (´run-hi < ´hi & ´array!´run-hi ≥ ´array!(´run-hi−1))
  DO ´run-hi :== ´run-hi+1 OD
 FI;;
 ´ret-value :== ´run-hi − ´lo;;
 ´ret :== ´array
CATCH Skip END
```

procedures (**imports** *globals-var*)
binary-sort(*array*::*int list*, *lo*::*nat*, *hi*::*nat*, *start*::*nat*| *ret*::*int list*)
where *pivot*::*int left*::*nat right*::*nat mid*::*nat move*::*nat* **in**

```
IF ´start = ´lo THEN ´start :== ´start + 1 FI;;
WHILE ´start < ´hi DO
 ´pivot :== ´array!´start;;
 ´left :== ´lo;;
 ´right :== ´start;;
 WHILE ´left < ´right DO
  ´mid :== (´left+´right) div 2;;
  IF ´pivot < ´array!´mid
  THEN
    ´right :== ´mid
  ELSE
    ´left :== ´mid+1
  FI
 OD;;
 ´move :== ´start − ´left;;
 ´array :== list-copy ´array (´left+1) ´array ´left ´move;;
 ´array!´left :== ´pivot;;
 ´start :== ´start + 1
OD;;
´ret :== ´array
```

procedures (**imports** *globals-var*)
sort(*array*::*int list*, *lo*::*nat*, *hi*::*nat*| *ret*::*int list*)
where *min-run*::*nat n-remaining*::*nat run-len-i*::*nat force*::*nat init-run-len-i*::*nat*
in

```
TRY
´n-remaining :== ´hi − ´lo;;
IF (´n-remaining < 2) THEN ´ret :== ´array;; THROW FI;;
IF (´n-remaining < 32) THEN
  CALL count-run-and-make-ascending(´array, ´lo, ´hi, ´init-run-len-i, ´array);;(∗
return two values∗)
  ´array :== CALL binary-sort(´array, ´lo, ´hi, ´lo+´init-run-len-i);;
  ´ret :== ´array;;
  THROW
FI;;
´a:==´array;;
´stack-size :== 0;;
´min-run :== 16;;
IF size ´a < 120 THEN ´stack-len :== 4 ELSE
IF size ´a < 1542 THEN ´stack-len :== 9 ELSE
IF size ´a < 119151 THEN ´stack-len :== 18 ELSE
´stack-len :== 39 FI FI FI;;
´run-len :== replicate ´stack-len (0::nat);;
```

```
´run-base :== replicate ´stack-len (0::nat);;
CALL count-run-and-make-ascending(´a, ´lo, ´hi, ´run-len-i, ´a);;
  IF ´run-len-i < ´min-run THEN
   IF ´n-remaining ≤ ´min-run THEN
    ´force :== ´n-remaining
   ELSE
    ´force :== ´min-run
   FI;;
   ´a :== CALL binary-sort(´a, ´lo, ´lo+´force, ´lo+´run-len-i);;
   ´run-len-i :== ´force
  FI;;
 CALL push-run(´lo, ´run-len-i);;
 CALL merge-collapse();;
 ´lo :== ´lo + ´run-len-i;;
 ´n-remaining :== ´n-remaining − ´run-len-i;;
 WHILE ´n-remaining ≠ 0 DO
 CALL count-run-and-make-ascending(´a, ´lo, ´hi ,´run-len-i, ´a);;
  IF ´run-len-i < ´min-run THEN
   IF ´n-remaining ≤ ´min-run THEN
    ´force :== ´n-remaining
   ELSE
    ´force :== ´min-run
   FI;;
   ´a :== CALL binary-sort(´a, ´lo, ´lo+´force, ´lo+´run-len-i);;
   ´run-len-i :== ´force
  FI;;
 CALL push-run(´lo, ´run-len-i);;
 CALL merge-collapse();;
 ´lo :== ´lo + ´run-len-i;;
 ´n-remaining :== ´n-remaining − ´run-len-i
 OD;;
CALL merge-force-collapse();;
´ret:==´a
CATCH Skip END
```

end