1 succsF approximates real control flow.

theory *JBC-succsFcorrect* = *JBC-SysInv*:

1.1 Auxiliary Definitions and Lemmas

lemma *catchstate-succsXpt*:
$\bigwedge p\ ps\ h\ stk\ loc\ e.\ [\![\ catchstate\ (P,X,(p,(None,h,(stk,loc,p)\#frs),e)) = (p',(None,h',(stk',loc',p')\#frs'),e');$
$match\text{-}ex\text{-}table\text{-}e\ P\ X\ (snd\ (snd\ p'))\ (ex\text{-}table\text{-}of\ P\ (fst\ p')\ (fst\ (snd\ p'))) = Some\ en;$
$pc\text{-}h = fst\ (snd\ (snd\ (snd\ en)));\ X' = fst\ (snd\ (snd\ en));$
$(fst\ p',fst\ (snd\ p'),pc\text{-}h) \in set\ (domC\ (P,An));$
$callers\text{-}sysinv\ ((P,An),(p,(None,h,(stk,loc,p)\#frs),e));$
$(\forall q{\in}set\ (p\#ps).\ match\text{-}ex\text{-}table\text{-}e\ P\ X\ (snd\ (snd\ q))\ (ex\text{-}table\text{-}of\ P\ (fst\ q)\ (fst\ (snd\ q))) = None)$
$]\!] \Longrightarrow$
$(\exists B.\ ((fst\ p',fst\ (snd\ p'),pc\text{-}h),B) \in set\ (succsXpt\ ((P,An),X,p\#ps)) \wedge$
$(B{=}TT \vee B{=}And\ [xcpt\text{-}cond\ (P,An)\ X\ (last\ (p\#ps))] \vee$
$B{=}\ And\ [Catch\ X'\ (aF\ (P,An)\ p'),Catch\ X\ (Pos\ p'),\ xcpt\text{-}cond\ (P,An)\ X\ (last\ (p\#ps))]))$

lemma *wf-match-ex-table-d*:
$[\![\ wf\ (P,An);\ (C,M) \in set\ (methodnames\ P);\ JVMExceptions.match\text{-}ex\text{-}table\ P\ X\ p\ (ex\text{-}table\text{-}of\ P\ C\ M) = Some\ (h,d)\ \ ]\!] \Longrightarrow d = 0$

lemma *findhandler-stk*:
$\bigwedge p.\ [\![\ wf\ \Pi;\ \forall p \in set\ (map\ (snd \circ snd)\ ((st',loc,p)\#frs)).\ p \in set\ (domC\ \Pi);\ \ find\text{-}handler\ (fst\ \Pi)\ xa\ h\ ((st',loc,p)\#frs) = (None,h,fr'\#frs')\ ]\!] \Longrightarrow fst\ fr' = [Addr\ xa]$

lemma *catchstate-Invoke*:
$\bigwedge s\ p\ e\ h\ fr\ fr'.\ [\![\ wf\ \Pi;\ s{=}(p,(None,h,fr\#fr'\#frs),e);\ s \in Reachables\ \Pi\ ]\!]{\Longrightarrow} (\exists M\ n.\ cmd\ \Pi\ (fst\ (catchstate\ (P,X,s))) = Some\ (Invoke\ M\ n))$

lemma *sys-xcpt-Reachables*:
$\bigwedge s\ p\ x\ h\ frs\ e.\ [\![\ s{=}(p,(x,h,frs),e);\ s \in Reachables\ \Pi;\ C \in sys\text{-}xcpts\ ]\!] \Longrightarrow (fst\ (the\ (h\ (addr\text{-}of\text{-}sys\text{-}xcpt\ C)))) = C$

lemma *callstates-Reachables*:
assumes *s-Reachables*: $s \in (Reachables\ \Pi)$
shows $\forall s' \in (callstates\ s).\ s' \in (Reachables\ \Pi)$ **using** *s-Reachables*

lemma *callstate-Reachables*:
$\bigwedge s.\ s \in (Reachables\ \Pi) \Longrightarrow callstate\ s \in (Reachables\ \Pi)$

lemma *catchstate-Reachables*:
$s \in Reachables\ \Pi \Longrightarrow catchstate(fst\ \Pi,X,s) \in (Reachables\ \Pi)$
lemma *match-ex-table-e-sim3*:
$(match\text{-}ex\text{-}table\text{-}e\ P\ C\ pc\ et = None) = (JVMExceptions.match\text{-}ex\text{-}table\ P\ C\ pc\ et = None)$
apply (*induct et*)
apply *simp*

apply *simp*
done

lemma *match-ex-table-e-matches-ex-entry*:
match-ex-table-e P C pc et = *Some e* ⟹ *matches-ex-entry P C pc e*
apply (*induct et*)
apply *simp*
apply (*simp split add*: *split-if-asm*)
done

lemma *match-ex-table-e-Some-X-X'*:
⟦ *match-ex-table-e P X pc et* = *Some* (*f*,*t*,*X'*,*pch*,*d*) ⟧ ⟹ *match-ex-table-e P X' pc et* = *Some* (*f*,*t*,*X'*,*pch*,*d*)
apply (*induct et*)
apply *simp*

apply (*simp add*: *matches-ex-entry-def split-def split add*: *split-if-asm*)
apply (*rule impI*)
apply (*erule conjE*)+
apply *simp*
apply (*drule match-ex-table-e-matches-ex-entry*)
apply (*drule match-ex-table-e-matches-ex-entry*)
apply (*simp add*: *matches-ex-entry-def*)
apply (*subgoal-tac* $P \vdash X \preceq^* $ *fst* (*snd* (*snd a*)))
 prefer *2*
 apply (*rule-tac b*=*X'* **in** *rtrancl-trans*)
 apply *simp*
 apply *simp*
apply *simp*
done

lemma *match-ex-table-e-None-X-X'*:
⟦ *match-ex-table-e P X pc et* = *None*; $P \vdash X \preceq^* X'$ ⟧ ⟹ *match-ex-table-e P X' pc et* = *None*
apply (*induct et*)
apply *simp*

apply (*simp add*: *matches-ex-entry-def split-def split add*: *split-if-asm*)
apply (*rule impI*)+
apply *simp*
apply (*rule classical*)
apply (*subgoal-tac* $P \vdash X \preceq^* $ *fst* (*snd* (*snd a*)))
 prefer *2*
 apply (*rule-tac b*=*X'* **in** *rtrancl-trans*)

```
 apply simp
 apply simp
apply simp
done

lemma catchstate-X-X':
assumes sigma-def: σ = (None, h, (stk, loc, p) # frs)
assumes p'-def: p' = (C',M',pc')
assumes σ'-def: σ' = (None, h', (stk', loc', p') # frs')
assumes catchstate-X: catchstate (P,X,(p,σ,e)) = (p',σ',e')
assumes match-ex-table-e-pc': match-ex-table-e P X pc' (ex-table-of P C' M') = Some (f,t,X',pch,d)
shows  catchstate (P, X', (p, σ , e)) = (p',σ',e')
proof –
from match-ex-table-e-pc'
have X-subcls-X': P ⊢ X ≼* X'
  apply –
  apply (drule match-ex-table-e-matches-ex-entry)
  apply (simp add: matches-ex-entry-def)
  done

have aux: ∀ σ stk loc (p::pos) (h::heap) (e::env). σ = (None,h,(stk,loc,p)#frs) ⟶
          catchstate (P,X,(p,σ,e)) = (p',σ',e') ⟶
          catchstate (P,X',(p,σ,e)) = (p',σ',e')
  proof (induct frs, intro allI impI)
    case Nil
    fix σ stk loc p h e
    assume sigma-def: (σ:: jvm-state) = (None, h, [((stk::val list), (loc::val list), (p::pos))])
    assume catchstate-X: catchstate (P, X, (p, σ, e)) = (p', σ', e')
    from sigma-def catchstate-X
    show catchstate (P, X', (p, σ, e)) = (p', σ', e')
      by fastsimp

  next

    case Cons
    fix fr frss
    assume hyp: ∀σ stk loc p h e.
             σ = (None, h, (stk, loc, p) # frss) ⟶
             catchstate (P, X, p, σ, e) = (p', σ', e') ⟶ catchstate (P, X', p, σ, e) = (p', σ', e')
    show ∀σ stk loc p h e.
             σ = (None, h, (stk, loc, p) # fr # frss) ⟶
             catchstate (P, X, p, σ, e) = (p', σ', e') ⟶ catchstate (P, X', p, σ, e) = (p', σ', e')
      proof (intro allI impI)
        fix σ stk loc p h e
        assume sigma-def: (σ::jvm-state) = (None, h, ((stk::val list), (loc::val list), (p::pos)) # fr #
```

```
frss)
        assume catchstate-X: catchstate (P, X, p, σ, e) = (p', σ', e')
        obtain fr-stk fr-loc fr-p
          where fr-def: fr = (fr-stk,fr-loc,fr-p)
          by (cases fr)
        obtain fr-C fr-M fr-pc
          where fr-p-def: fr-p = (fr-C,fr-M,fr-pc)
          by (cases fr-p)
        show catchstate (P, X', p, σ, e) = (p', σ', e')
          proof (cases match-ex-table-e P X fr-pc (ex-table-of P fr-C fr-M))
           case None
           note X-None = None
           from X-None X-subcls-X'
           have X'-None:
             match-ex-table-e P X' fr-pc (ex-table-of P fr-C fr-M) = None
             by (rule match-ex-table-e-None-X-X')

           from X-None X'-None catchstate-X fr-def fr-p-def sigma-def
           show ?thesis
             apply -
             apply (simp add: match-ex-table-e-sim3)
             apply (cut-tac hyp)
             apply (erule-tac x=(None,hd (cs e),(fr-stk, fr-loc, fr-C, fr-M, fr-pc) # frss) in allE)
             apply (erule-tac x=fr-stk in allE)
             apply (erule-tac x=fr-loc in allE)
             apply (erule-tac x=fr-p in allE)
             apply (erule-tac x=hd (cs e) in allE)
             apply (erule-tac x=e(|cs := tl (cs e)|) in allE)
             apply simp
             done

          next

           case Some

           fix en'
           assume X-en': match-ex-table-e P X fr-pc (ex-table-of P fr-C fr-M) = ⌊en'⌋
           show catchstate (P, X', p, σ, e) = (p', σ', e')
             proof -

               from X-en'
               obtain pcd
                    where X-pcd: JVMExceptions.match-ex-table P X fr-pc (ex-table-of P fr-C fr-M) =
⌊pcd⌋
                 and pcd-def: pcd = snd (snd (snd en'))
```

```
          apply −
          apply (drule match-ex-table-e-sim)
          by fastsimp

        from X-en′ X-pcd pcd-def sigma-def catchstate-X fr-def fr-p-def
             σ′-def p′-def match-ex-table-e-pc′
        show ?thesis
          apply −
          apply (simp split del: option.split-asm)
          apply (drule-tac t=en′ in sym)
          apply simp
          apply (drule match-ex-table-e-Some-X-X′)
          apply (drule match-ex-table-e-sim)
          apply simp
          done
      qed
    qed
  qed
qed
from aux catchstate-X sigma-def
show ?thesis
  apply −
  apply (erule-tac x=σ in allE)
  apply (erule-tac x=stk in allE)
  apply (erule-tac x=loc in allE)
  apply (erule-tac x=p in allE)
  apply (erule-tac x=h in allE)
  apply (erule-tac x=e in allE)
  by fastsimp
qed

lemma findhandler-succsXpt:
assumes wf-Pi: wf Π
assumes s-ReachablesAn:s ∈ ReachablesAn Π
assumes s-def: (s::jbc-state) = (p,(None,h,(stk,loc,p)#frs),e)
assumes p-def:  p = (C,M,pc)
assumes cmd-p: cmd Π p = Some i
assumes check-s: check (fst Π) (None,h,(stk,loc,p)#frs)
assumes exec-i: exec-instr i (fst Π) h stk loc C M pc frs = (Some xa,h′,frs′′)
assumes find-handler-s: find-handler (fst Π) xa h ((stk,loc,p)#frs) = (None,h,fr′#frs′)
assumes X-def: fst (the (h xa)) = X
shows ∃B. (snd (snd (fr′)),B) ∈ set (succsXpt (Π,X,[p])) ∧ Π,s ⊨ B
proof −
from s-ReachablesAn
have s-Reachables: s ∈ Reachables Π
```

by (*rule ReachablesAn-Reachables*)

from *wf-Pi s-Reachables s-def*
have *frs-domC*: ∀ *p*∈*set* (*map* (*snd* ∘ *snd*) ((*stk*, *loc*, *p*) # *frs*)). *p* ∈ *set* (*domC* Π)
apply (*rule-tac p*=*p* **and** *x*=*None* **and** *h*=*h* **and**
frs=(*stk*,*loc*,*p*)#*frs* **and** *e*=*e* **in** *wf-Reachables-domC′*)
apply *assumption*
apply *simp*
done

from *cmd-p*
have *p-domC*: *p* ∈ *set* (*domC* Π)
by (*rule cmd-domC*)

from *wf-Pi find-handler-s frs-domC*
have *fst-fr′*: *fst fr′* = [*Addr xa*]
apply (*frule-tac xa*=*xa* **and** *h*=*h* **and** *st′*=*stk* **and** *loc*=*loc* **and** *p*=*p*
and *frs*=*frs* **and** *fr′*=*fr′* **and** *frs′*=*frs′* **in** *findhandler-stk*)
by *assumption+*

from *fst-fr′*
obtain *loc′ C′ M′ pc′* **where** *fr′-def*: *fr′*=([*Addr xa*],*loc′*,*C′*,*M′*,*pc′*)
apply (*cases fr′*)
by *fastsimp*

obtain *P An* **where** *Pi-def*: Π = (*P*,*An*)
by (*cases* Π)

from *wf-Pi Pi-def cmd-p p-domC p-def*
have *i-instrs-of*: (*instrs-of P C M* ! *pc*) = *i*
apply −
apply (*drule domC-cmd-instr-of*)
apply *simp*
apply *simp*
done

from *find-handler-s p-def* **obtain** *pfx*
where *pfx-def*: (*stk*, *loc*, *C*, *M*, *pc*) # *frs* = *pfx* @ *frs′*
and *pfx-ne-Nil*: *pfx* ≠ []
apply −
apply (*drule find-handler-frs*)
by *fastsimp*

show *?thesis*

proof (*cases frs* = *frs'*)

case *True*

note *frs-eq-frs'* = *True*

show *?thesis*
proof (*cases JVMExceptions.match-ex-table P X pc* (*ex-table-of P C M*))

case *None*
 from *None frs-eq-frs' find-handler-s s-def p-def Pi-def X-def*
 show *?thesis*
   apply *simp*
   apply (*drule find-handler-frs*)
   apply *simp*
   done

next

case (*Some pcd*)

from *Some*
obtain *en*
where *en-intro*: *match-ex-table-e P X pc* (*ex-table-of P C M*) = *Some en*
and *en-pcd*: *snd* (*snd* (*snd en*)) = *pcd*
 apply −
 apply (*drule match-ex-table-e-sim2*)
 apply (*erule exE* | *erule conjE*)+
 by *fastsimp*

from *en-pcd*
obtain *f t X'*
where *en-def*: *en* = (*f*,*t*,*X'*,*pcd*)
  apply (*cases en*)
  by *fastsimp*

from *p-domC p-def Pi-def*
have *C-M-methodnames*: (*C*,*M*) ∈ *set* (*methodnames P*)
  by (*rule-tac pc*=*pc* **and** *An*=*An* **in** *domC-methodnames*,*simp*)

from *frs-eq-frs' Some find-handler-s s-def fr'-def X-def Pi-def p-def*
obtain *d* **where** *CeqC'*: *C* = *C'*
  and *MeqM'*: *M* = *M'*
  and *pcd-pc'*: *pcd* = (*pc'*,*d*)
  apply (*cases pcd*)

by *fastsimp*

from *Some wf-Pi C-M-methodnames Pi-def p-def CeqC′ MeqM′ pcd-pc′*
have *p′-domC*: (*C′*,*M′*,*pc′*) ∈ *set* (*domC* (*P*,*An*))
apply −
apply (*rule-tac X*=*X* **and** *pc*=*pc* **and** *pc′*=*pc′* **in** *wf-ex-table-domC*)
apply *simp*+
done

show *?thesis*
proof (*cases length* (*domC* (*P*, *An*)) ≤ *Suc 0*)

case *True*
from *True p′-domC fr′-def Pi-def*
show *?thesis*
apply −
apply (*rule-tac x*=*TT* **in** *exI*)
apply *simp*
done

next

case *False*

note *domC-length* = *False*

note *mainsimps* = *en-intro en-def domC-length Pi-def cmd-p exec-i*
p-def fr′-def pcd-pc′ CeqC′ MeqM′ xcpt-cond-def s-def
check-s i-instrs-of
show *?thesis*
proof (*cases i*)

case (*Load n*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*Store n*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*Push v*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
  by *simp*

next

case (*New Cl*)
from *this mainsimps*
show *?thesis*
  by *simp*

next

case (*Getfield F C*)
from *this mainsimps*
show *?thesis*
  apply −
  apply (*case-tac stk*)
  apply (*simp add*: *split-def check-def split add*: *split-if-asm*)
  apply (*simp add*: *split-def split add*: *split-if-asm*)
  done

next

case (*Putfield F C*)
from *this mainsimps*
show *?thesis*
  apply −
  apply (*case-tac stk*)
  apply (*simp add*: *split-def check-def split add*: *split-if-asm*)

  apply (*simp add*: *split-def hd-list-def split add*: *split-if-asm*)
  apply (*case-tac list*)
  apply (*simp add*: *split-def check-def*)
  apply *simp*
  done

next

case (*Checkcast Cl*)

show *?thesis*
  proof (*cases cast-ok P Cl h* (*hd stk*))

case *True*
from *True Checkcast mainsimps*
show *?thesis*
  by *simp*

next

case *False*
from *Checkcast s-def check-s Pi-def i-instrs-of p-def*
obtain *r stks*
where *stk-def*: *stk* = *r#stks*
and *r-isRef'*: *is-Ref' h r*
and *is-class-Cl*: *is-class P Cl*
  apply (*cases stk*)
  apply (*simp add*: *check-def split-def*)
  apply (*simp add*: *check-def split-def*)
  apply *fastsimp*
  done

from *False stk-def r-isRef' is-Ref'-def*
obtain r' **where** *r-Addr*: $r = Addr\ r' \wedge h\ r' \neq None$
  apply −
  apply (*simp add*: *cast-ok-def*)
  apply (*case-tac r*::*val*)
  apply *simp*+
  apply *fastsimp*
  done

from *False Checkcast mainsimps stk-def r-Addr is-class-Cl is-Ref'-def*
show *?thesis*
  apply −
  apply (*simp add*: *evalEs-map cast-ok-def list-all-iff*)
  apply (*rule allI*)+
  apply (*rule impI*)+
  apply (*case-tac x* = *a*)
  apply (*erule conjE* | *erule exE*)+
  apply *simp*

  apply *simp*
  done
qed

next
case (*Invoke Mn n*)
from *this mainsimps*

show *?thesis*
apply (*case-tac stk* ! *n* = *Null*)
apply (*simp add*: *check-def split-def*)

apply (*simp add*: *split-def*)
done
next
case *Return*
from *this mainsimps*
show *?thesis*
by *simp*

next

case *Pop*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case *IBin*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*Goto t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case *CmpEq*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*IfIntCmp ro t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*

by *simp*

next

case (*IfFalse t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next
case *Throw*

from *Throw s-def check-s Pi-def i-instrs-of p-def*
obtain *v stks*
where *stk-def*: $stk = v\#stks$
and *v-isRef'-Null*: $is\text{-}Ref'\ h\ v \vee v = Null$
apply (*cases stk*)
apply (*simp add*: *check-def split-def*)
apply (*simp add*: *check-def split-def*)
apply *fastsimp*
done

from *stk-def v-isRef'-Null is-Ref'-def*
obtain *r* **where** *v-Addr*: $v \neq Null \longrightarrow (v = Addr\ r \wedge h\ r \neq None)$
apply −
apply (*case-tac v*::*val*)
apply *simp*+
apply *fastsimp*
done

from *s-Reachables s-def sys-xcpts-def*
have *cname-NullPointer*: *cname-of h* (*addr-of-sys-xcpt NullPointer*) = *NullPointer*
apply −
apply (*rule sys-xcpt-Reachables*)
apply *assumption*
apply *assumption*
apply *simp*
done

from *Throw mainsimps stk-def v-Addr X-def cname-NullPointer*
show *?thesis*
apply −
apply *simp*
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=frs''* **in** *sym*)

apply *simp*
apply (*case-tac v* = *Null*)
apply *simp*

apply *simp*
apply *fastsimp*
done
qed
qed
qed

next
— frs = frs'
case *False*
note *frs-neq-frs'* = *False*

from *frs-neq-frs' s-def fr'-def find-handler-s Pi-def p-def X-def*
obtain *hc stkc pcc* **where** *catchstate-s*:
catchstate (*P*, *X*, *p*, (*None*, *h*, (*stk*, *loc*, *p*) # *frs*), *e*) =
((*C'*, *M'*, *pcc*), (*None*, *hc*, (*stkc*, *loc'*, *C'*, *M'*, *pcc*) # *frs'*), *e*(|*cs* := *drop* (*length frs* − *length frs'*) (*cs* *e*)|))
and *match-ex-table-pc'*:*JBC-VCG.match-ex-table P X pcc* (*ex-table-of P C' M'*) = ⌊*pc'*⌋
apply −
apply (*simp only*:)
apply (*frule-tac stk*=*stk* **and** *e*=*e* **in** *find-handler-catchstate'*)
apply (*simp only*: *simp-thms*)
apply (*erule exE* | *erule conjE*)+
apply *fastsimp*
done

from *match-ex-table-pc'*
obtain *pcd* **where** *pcd-intro*: *JVMExceptions.match-ex-table P X pcc* (*ex-table-of P C' M'*) = ⌊*pcd*⌋
and *pcd-pc'*: *fst pcd* = *pc'*
apply −
apply (*rule classical*)
apply *simp*
apply *fastsimp*
done

from *pcd-pc'*
obtain *d* **where** *pcd-def*: *pcd* = (*pc'*,*d*)
apply (*cases pcd*)
by *fastsimp*

from *pcd-intro*

```
obtain en
where en-intro: match-ex-table-e P X pcc (ex-table-of P C′ M′) = Some en
and en-pcd: snd (snd (snd en)) = pcd
 apply −
 apply (drule match-ex-table-e-sim2)
 apply (erule exE | erule conjE)+
 by fastsimp

from en-pcd
obtain f t X′
where en-def: en = (f,t,X′,pcd)
  apply (cases en)
  by fastsimp

from pcd-intro Pi-def p-def
have p′-domC: (C′,M′,pc′) ∈ set (domC Π)
  proof −
    from frs-domC find-handler-s s-def fr′-def Pi-def
    have methodnames-C′M′: (C′,M′) ∈ set (methodnames P)
      apply −
      apply (drule find-handler-frs′)
      apply (erule conjE | erule exE)+
      apply simp
      apply (rule-tac xs=pfx in rev-cases)
      apply simp

      apply (simp only: last-snoc)
      apply (subgoal-tac ∃ y1 y2 y3 y4 y5. y = (y1,y2,y3,y4,y5))
       prefer 2
       apply fastsimp
      apply (erule exE)+
      apply (simp add: split-def)
      apply (erule conjE)+
      apply (rule-tac An=An and pc=snd (snd (snd (snd y))) in domC-methodnames)
      apply simp
      done
    from wf-Pi pcd-intro methodnames-C′M′ pcd-def Pi-def
    show ?thesis
      apply −
      apply (simp only:)
      apply (drule wf-ex-table-domC)
      apply assumption
      apply assumption
      apply assumption
      done
```

qed

from *wf-Pi s-Reachables*
have *callers-sysinv-s*: *callers-sysinv* (Π,*s*)
by (*rule callers-sysinv-Reachables*)

from *frs-neq-frs′ find-handler-s s-def p-def Pi-def X-def*
have *no-match-ex-table-e-p*: *match-ex-table-e P X pc* (*ex-table-of P C M*) = *None*
apply −
apply *simp*
apply (*rule classical*)
apply *simp*
apply (*erule exE*)+
apply (*drule match-ex-table-e-sim*)
apply *simp*
done

from *catchstate-s en-intro en-pcd en-def pcd-def p′-domC callers-sysinv-s s-def no-match-ex-table-e-p Pi-def p-def X-def*
obtain *B*
where *p′-B-succsXpt*: ((*C′*,*M′*,*pc′*), *B*) ∈ *set* (*succsXpt* (Π, *X*, [*p*]))
and *B-def*:*B* = *TT* ∨ *B* = *And* [*xcpt-cond* (*P*,*An*) *X* (*last* [*p*])] ∨
B= *And* [*Catch X′* (*aF* Π (*C′*,*M′*,*pcc*)), *Catch X* (*Pos* (*C′*,*M′*,*pcc*)), *xcpt-cond* Π *X p*]
apply −
apply (*drule-tac P*=*P* **and** *An*=*An* **and** *X*=*X* **and** *X′*=*X′* **and** *p*=*p* **and** *h*=*h* **and** *stk*=*stk* **and** *loc*=*loc*
and *frs*=*frs* **and** *e*=*e* **and** *h′*=*hc* **and** *stk′*=*stkc* **and** *loc′*=*loc′* **and** *p′*=(*C′*,*M′*,*pcc*)
and *frs′*=*frs′* **and** *e′*=*e*⦇*cs*:=*drop* (*length frs* − *length frs′*) (*cs e*)⦈ **and** *ps*=[] **in** *catchstate-succsXpt*)
apply *simp*
apply *simp*
apply *simp*
apply *simp*
apply *simp*
apply *simp*
apply (*erule exE* | *erule conjE*)+
apply *fastsimp*
done

obtain *σ′ e′ p′*
where *catchstate-s-def2*: *catchstate* (*P*,*X*,*s*) = (*p′*,*σ′*,*e′*)
by (*cases catchstate* (*P*,*X*,*s*))

from *catchstate-s s-def catchstate-s-def2*
have *p′-def*: *p′* = (*C′*,*M′*,*pcc*)
and *σ′-def*: *σ′* = (*None*, *hc*, (*stkc*, *loc′*, *C′*, *M′*, *pcc*) # *frs′*)

and *e′-def*: *e′* = *e*⦇*cs* := *drop* (*length frs* − *length frs′*) (*cs e*)⦈
apply −
apply *fastsimp*+
done

from *s-def catchstate-s-def2 p′-def σ′-def e′-def en-intro en-pcd pcd-def en-def p′-def*
have *catchstate-s-X-X′*:
catchstate (*P*,*X′*,*s*) = (*p′*,*σ′*,*e′*)
apply −
apply (*simp only*:)
apply (*rule-tac h*=*h* **and** *stk*=*stk* **and** *loc*=*loc* **and** *frs*=*frs* **and** *pch*=*pc′*
and *C′*=*C′* **and** *M′*=*M′* **and** *pc′*=*pcc* **and** *h′*=*hc* **and** *stk′*=*stkc*
and *loc′*=*loc′* **and** *frs′*=*frs′* **and** *X*=*X* **and** *f*=*f* **and** *t*=*t* **and** *d*=*d*
in *catchstate-X-X′*)
apply (*rule refl*)
apply (*rule refl*)
apply (*rule refl*)
apply *assumption*
apply *simp*
done

from *frs-neq-frs′ pfx-def pfx-ne-Nil*
obtain *fr2 frs2* **where** *frs-def*: *frs* = *fr2*#*frs2*
apply −
apply (*case-tac frs*)
apply (*simp add*: *neq-Nil-conv*)
apply (*erule exE*)+
apply *simp*

apply *fastsimp*
done

have *catchstate-s-aF*: Π,(*p′*,*σ′*,*e′*) ⊨ (*aF* Π *p′*)
proof −
from *Pi-def s-ReachablesAn catchstate-s-X-X′*
have *catchstate-s-ReachablesAn*: (*p′*,*σ′*,*e′*) ∈ *ReachablesAn* Π
apply −
apply (*drule-tac X*=*X′* **in** *catchstate-ReachablesAn*)
apply *simp*
done

from *wf-Pi*
have *ipc-domC*: *ipc* Π ∈ *set* (*domC* Π)
by (*rule wf-ipc-domC*)

from *wf-Pi ipc-domC catchstate-s-ReachablesAn Pi-def catchstate-s-X-X'*
have *catchstate-s-initF-aF*: Π,$(p',\sigma',e') \models$ *initF* $\Pi \vee \Pi$,$(p',\sigma',e') \models$ *aF* Π p'
apply −
apply (*rule ReachablesAn-initF-aF*)
apply (*simp add*: *mem-iff*)+
done

show *?thesis*
proof (*cases* Π,$(p',\sigma',e') \models$ *initF* Π)

case *True*
note *catchstate-s-initF* = *True*

from *catchstate-s-initF*
have *p'-ipc*: p' = *ipc* Π
by (*simp add*: *initF-def*)

from *wf-Pi s-def frs-def s-Reachables Pi-def p-def catchstate-s-X-X'*
obtain *Mn n*
where *cmd-p'*: *cmd* Π p' = *Some* (*Invoke Mn n*)
apply −
apply (*drule-tac X*=*X'* **and** *P*=*P* **in** *catchstate-Invoke*)
apply *simp*
apply *simp*
apply *fastsimp*
done

from *cmd-p' p'-ipc wf-Pi*
show *?thesis*
apply −
apply (*drule wf-ipc-no-Invoke*)
apply *simp*
done

next

case *False*

from *False catchstate-s-X-X' catchstate-s-initF-aF p'-def*
show *?thesis*
by *simp*
qed
qed

from *Pi-def s-Reachables*

have *catchstate-s-Reachables*: *catchstate* $(P, X', s) \in$ *Reachables* Π
  apply –
  apply (*drule catchstate-Reachables*)
  apply *simp*
  done

from *wf-Pi catchstate-s catchstate-s-def2 catchstate-s-X-X′ catchstate-s-Reachables*
have *catchstate-s-Pos-p′*: $\Pi,(p',\sigma',e') \models$ *Pos* p'
  apply –
  apply (*drule-tac* $s=(p',\sigma',e')$ **in** *inv-Pos-Reachable*)
  apply *simp*
  apply (*simp add*: *inv-Pos-def*)
  done

note *mainsimps* = *en-intro en-def Pi-def cmd-p exec-i*
    p-def fr′-def pcd-pc′ p′-def xcpt-cond-def s-def
    check-s i-instrs-of

have *s-xcpt-cond*: $\Pi, s \models$ *xcpt-cond* Π X p
proof (*cases i*)

case (*Load n*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
  by *simp*

next

case (*Store n*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
  by *simp*

next

case (*Push v*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
  by *simp*

next

case (*New Cl*)
from *this mainsimps*
show *?thesis*

by *simp*

next

case (*Getfield F C*)
from *this mainsimps*
show *?thesis*
apply −
apply (*case-tac stk*)
apply (*simp add*: *split-def check-def split add*: *split-if-asm*)
apply (*simp add*: *split-def split add*: *split-if-asm*)
done

next

case (*Putfield F C*)
from *this mainsimps*
show *?thesis*
apply −
apply (*case-tac stk*)
apply (*simp add*: *split-def check-def split add*: *split-if-asm*)

apply (*simp add*: *split-def hd-list-def split add*: *split-if-asm*)
apply (*case-tac list*)
apply (*simp add*: *split-def check-def*)
apply *simp*
done

next

case (*Checkcast Cl*)

show *?thesis*
proof (*cases cast-ok P Cl h* (*hd stk*))
case *True*
from *True Checkcast mainsimps*
show *?thesis*
by *simp*

next

case *False*
from *Checkcast s-def check-s Pi-def i-instrs-of p-def*
obtain *r stks*
where *stk-def*: *stk* = *r#stks*

and *r-isRef'*: *is-Ref' h r*
and *is-class-Cl*: *is-class P Cl*
apply (*cases stk*)
apply (*simp add*: *check-def split-def*)
apply (*simp add*: *check-def split-def*)
apply *fastsimp*
done

from *False stk-def r-isRef' is-Ref'-def*
obtain r' **where** *r-Addr*: $r = Addr\ r' \wedge h\ r' \neq None$
apply −
apply (*simp add*: *cast-ok-def*)
apply (*case-tac r*::*val*)
apply *simp*+
apply *fastsimp*
done

from *False Checkcast mainsimps stk-def r-Addr is-class-Cl is-Ref'-def*
show *?thesis*
apply −
apply (*simp add*: *evalEs-map cast-ok-def list-all-iff*)
apply (*rule allI*)+
apply (*rule impI*)+
apply (*case-tac* $x = a$)
apply (*erule conjE* | *erule exE*)+
apply *simp*

apply *simp*
done
qed

next
case (*Invoke Mn n*)
from *this mainsimps*
show *?thesis*
apply (*case-tac stk* ! $n = Null$)
apply (*simp add*: *check-def split-def*)

apply (*simp add*: *split-def*)
done
next
case *Return*
from *this mainsimps*
show *?thesis*
by *simp*

next

case *Pop*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case *IBin*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*Goto t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case *CmpEq*
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*IfIntCmp ro t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next

case (*IfFalse t*)
from *this Pi-def cmd-p exec-i*
show *?thesis*
by *simp*

next
case *Throw*

from *Throw s-def check-s Pi-def i-instrs-of p-def*
obtain *v stks*
where *stk-def*: *stk* = *v#stks*
and *v-isRef′-Null*: *is-Ref′ h v* ∨ *v* = *Null*
apply (*cases stk*)
apply (*simp add*: *check-def split-def*)
apply (*simp add*: *check-def split-def*)
apply *fastsimp*
done

from *stk-def v-isRef′-Null is-Ref′-def*
obtain *r* **where** *v-Addr*: *v* ≠ *Null* ⟶ (*v* = *Addr r* ∧ *h r* ≠ *None*)
apply −
apply (*case-tac v*::*val*)
apply *simp+*
apply *fastsimp*
done

from *s-Reachables s-def sys-xcpts-def*
have *cname-NullPointer*: *cname-of h* (*addr-of-sys-xcpt NullPointer*) = *NullPointer*
apply −
apply (*rule sys-xcpt-Reachables*)
apply *assumption*
apply *assumption*
apply *simp*
done

from *Throw mainsimps stk-def v-Addr X-def cname-NullPointer*
show *?thesis*
apply −
apply *simp*
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=frs″* **in** *sym*)
apply *simp*
apply (*case-tac v* = *Null*)
apply *simp*

apply *simp*
apply *fastsimp*
done
qed

from *p′-B-succsXpt B-def catchstate-s-X-X′ catchstate-s-def2 catchstate-s-Pos-p′ p′-def catchstate-s-aF fr′-def s-xcpt-cond Pi-def s-def frs-def*

show *?thesis*
  apply −
  apply (*erule disjE*)
  apply (*rule-tac x*=*TT* **in** *exI*)
  apply *simp*

  apply (*erule disjE*)
  apply (*rule-tac x*=*And* [*xcpt-cond* (*P*, *An*) *X* (*last* [*p*])] **in** *exI*)
  apply *simp*

  apply (*rule-tac x*=*And* [*Catch X'* (*aF* Π *p'*), *Catch X* (*Pos p'*), *xcpt-cond* Π *X p*] **in** *exI*)
  apply (*simp del*: *succsXpt.simps*)
  done
qed
qed

1.2 succsF correct

lemma *succsF-correct*:
assumes *wf-Pi*:*wf* Π
assumes *in-ReachablesAn*:*s* ∈ (*ReachablesAn* Π)
assumes *s-s'-effS*: (*s*,*s'*) ∈ (*effS* Π)
shows (∃ *B*. (*fst s'*,*B*) ∈ *set* (*succsF* Π (*fst s*)) ∧ Π,*s* ⊨ *B*)
end