1 Branch conditions guarantee progress

theory *JBC-succsFprogress* = *JBC-VCG-Correctness*:

lemma *catchstate-find-handler*:
$\bigwedge$ *st rg C M pc e stk*. $[\![$
wf (*P*,*An*);
$\forall$ *p* $\in$ *set* (*map* (*snd* $\circ$ *snd*) ((*st*,*rg*,(*C*,*M*,*pc*))#*frs*)). *p* $\in$ *set* (*domC* (*P*,*An*)); (C',M',pc') $\in$ *set* (*domC* (*P*,*An*));
catchstate (*P*,(*cname-of h xa*),((*C*,*M*,*pc*),(*None*,*h*,(*st*,*rg*,(*C*,*M*,*pc*))#*frs*),*e*)) =
((C',M',pc'),(x',h',(st',rg',(C',M',pc'))#frs'),e');
JVMExceptions.match-ex-table P (*cname-of h xa*) pc' (*ex-table-of P* C' M') = *Some* (*pch*,*d*);
JVMExceptions.match-ex-table P (*cname-of h xa*) *pc* (*ex-table-of P C M*) = *None*
$]\!] \Longrightarrow$ *find-handler P xa h* ((*stk*,*rg*,(*C*,*M*,*pc*))#*frs*) = (*None*,*h*,([*Addr xa*],rg',(C',M',*pch*))#frs')

consts *callchain*::*jbc-prog* $\Rightarrow$ *pos list* $\Rightarrow$ *bool*
primrec
callchain Π [] = *True*
callchain Π (*p*#*ps*) = (*case ps of* [] $\Rightarrow$ *True*
| p'#ps' $\Rightarrow$ *p* $\in$ *set* (*callers* Π p') $\wedge$ *callchain* Π *ps*)

lemma *callchain-append*:
$\bigwedge$ *p* ps'. *callchain* Π (*ps*@*p*#ps') = (*callchain* Π (*ps*@[*p*]) $\wedge$ *callchain* Π (*p*#ps'))

lemmas *succsXpt-def* = *succsXpt.simps*

lemma *succsXpt-simp*: *succsXpt* ((*P*, *An*), *X*, *ps*) =
(*case length* (*domC* (*P*, *An*)) $\leq$ *length ps of True* $\Rightarrow$ *map* (λ*p*. (*p*, *TT*)) (*domC* (*P*, *An*))
| *False* $\Rightarrow$ *case ps of* [] $\Rightarrow$ *map* (λ*p*. (*p*, *TT*)) (*domC* (*P*, *An*))
| *p* # *pss* $\Rightarrow$
let (*C*, *M*, *pc*) = *p*; *et* = *ex-table-of P C M*; *A* = *aF* (*P*, *An*) *p*
in (*case match-ex-table-e P X pc et of None* $\Rightarrow$ *concat* (*map* ($\lambda p'$. *succsXpt* ((*P*, *An*), *X*, p' # *ps*)) (*callers* (*P*, *An*) *p*))
| $\lfloor en \rfloor$ $\Rightarrow$ (*let* (*f*,*t*,X',pc',*d*) = *en*
in [((*C*, *M*, pc'), *And* ((*if pss* = [] *then* []
else [*Catch* X' *A*, *Catch X* (*Pos p*)])@[*xcpt-cond* (*P*, *An*) *X* (*last ps*)]))]))))

lemma *catchstate-form'*:
$\bigwedge$ *p h e stk loc frs*. $\exists$ p' h' st' rg' frs' e'. *catchstate* (*P*,*X*,(*p*,(*None*,*h*,(*stk*,*loc*,*p*)#*frs*),*e*)) = (p',(*None*,h',(st',rg',p')#frs')

lemma *succsXpt-findhandler*:
assumes *s-def*:$s = (p,\ (None,\ h,\ (st,\ rg,\ p)\ \#\ frs),\ e)$
assumes *wf-Pi*:*wf* Π
assumes p'*-B-succsXpt*:$(p',B) \in set\ (succsXpt\ (\Pi,X,[p]))$
assumes *valid-B*:$\Pi,s \models B$
assumes *s-inv-Pos*: $\Pi,s \models inv\text{-}Pos\ \Pi\ (fst\ s)$
assumes *X-def*: $X = cname\text{-}of\ h\ xa$
assumes *succsXpt-domC*: $fst\ `\ set\ (succsXpt\ (\Pi,\ X,\ [p])) \subset set\ (domC\ \Pi)$
shows
$\exists\ rg'\ frs'.\ find\text{-}handler\ (fst\ \Pi)\ xa\ h\ ((st,rg,p)\#frs) = (None,h,([Addr\ xa],rg',p')\#frs')$
proof −

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

obtain *C M pc*
where *p-def*:$p = (C,M,pc)$
  by (*cases p*)

from *s-inv-Pos* **have**
callers-sysinv-s:*callers-sysinv* (Π,s)
  apply −
  apply (*rule callers-sysinv-Pos*)
  by (*simp add*: *inv-Pos-def*)

from *callers-sysinv-s s-def*
obtain *dm dm'* **where** *domC-p*:$domC\ \Pi = dm@[p]@dm'$
  apply −
  apply (*simp only*: *callers-sysinv-trans*)
  apply (*erule-tac x=0* **in** *allE*)
  apply (*simp add*: *in-set-conv-decomp*)
  apply (*erule conjE* | *erule exE*)+
  by *fastsimp*

have *succsXpt-induct*:
  $\bigwedge k\ L.\ length\ (domC\ \Pi) - length\ L = k$
  $\Longrightarrow fst\ `\ set\ (succsXpt\ (\Pi,\ X,\ L)) \subset set\ (domC\ \Pi)$
  $\Longrightarrow (\exists L'.\ L = L'\ @\ [p])$
  $\Longrightarrow (p',\ B) \in set\ (succsXpt\ (\Pi,\ X,\ L))$
  $\Longrightarrow (\forall (C',\ M',\ pc')\in set\ (tl\ L).\ JBC\text{-}VCG.match\text{-}ex\text{-}table\ P\ X\ pc'\ (ex\text{-}table\text{-}of\ P\ C'\ M') = None)$
  $\Longrightarrow callchain\ \Pi\ L \Longrightarrow$
  $\exists\ rg'\ frs'.\ find\text{-}handler\ P\ xa\ h\ ((st,rg,p)\#frs) = (None,h,([Addr\ xa],rg',p')\#frs')$ (**is** $\bigwedge k.$ *PROP ?P k*)

proof −
fix k **show** *PROP ?P k*
proof (*induct k rule*: *nat-less-induct*)
case (*1 n*)
assume *IH*:$\forall m<n. \forall x.$ *length* (*domC* Π) − *length* $x = m \longrightarrow$
fst ' *set* (*succsXpt* (Π, X, x)) $\subset$ *set* (*domC* Π) $\longrightarrow$
($\exists L'. x = L'$ @ [p]) $\longrightarrow$
(p', B) $\in$ *set* (*succsXpt* (Π, X, x)) $\longrightarrow$
($\forall$ (C', M', pc')$\in$*set* (*tl x*). *JBC-VCG.match-ex-table P X pc'* (*ex-table-of P C' M'*) = *None*) $\longrightarrow$
callchain Π $x \longrightarrow$ ($\exists$ *rg' frs'*. *find-handler P xa h* ((*st*,*rg*,*p*)#*frs*) = (*None*,*h*,([*Addr xa*],*rg'*,*p'*)#*frs'*))

assume *length-L-n*:*length* (*domC* Π) − *length* $L = n$
assume *succsX-domC*:*fst* ' *set* (*succsXpt* (Π, X, L)) $\subset$ *set* (*domC* Π)
assume *appL-L'*:$\exists L'. L = L'$ @ [p]
assume *p'-B-succsX-L*:(p', B) $\in$ *set* (*succsXpt* (Π, X, L))
assume *match-ex-tab-L*:$\forall$ $u$$\in$*set* (*tl L*). ($\lambda$($C'$, M', pc'). *JBC-VCG.match-ex-table P X pc'* (*ex-table-of P C' M'*) = *None*) u
assume *callchain-L*: *callchain* Π L
show $\exists$ *rg' frs'*. *find-handler P xa h* ((*st*,*rg*,*p*)#*frs*) = (*None*,*h*,([*Addr xa*],*rg'*,*p'*)#*frs'*)
proof −

from *succsX-domC* **have** *domC-L*:$\neg$ (*length* (*domC* Π) $\leq$ *length* L)
by (*simp add*: *succsXpt.simps Pi-def split add*: *bool.split-asm*)

from *succsX-domC domC-L appL-L'* **obtain** *Ca Ma pca list* **where** *L-cons*: L = (*Ca*,*Ma*,*pca*)#*list*
apply (*case-tac* L = [])
apply *simp*
apply (*simp add*: *neq-Nil-conv*, (*erule exE*)+,*blast*)
done

show *?thesis*
proof (*cases JVMExceptions.match-ex-table P X pca* (*ex-table-of P Ca Ma*))
case *None*
from *None domC-L L-cons p'-B-succsX-L Pi-def*
have *succsXpt-rec*: ($\exists$ $a$$\in$*set* (*callers* Π (*Ca*, *Ma*, *pca*)). (p',B) $\in$ *set* (*succsXpt* (Π, X, a # (*Ca*, *Ma*, *pca*) # *list*)))
apply (*simp only*: *Pi-def*)
apply (*drule-tac* P=λ S. (p', B) $\in$ *set* S **in** *subst*[*OF succsXpt-simp*])
apply (*simp del*: *succsXpt.simps split del*: *option.split-asm*)
apply (*simp add*: *set-concat-map split-def del*: *succsXpt.simps*)
apply (*drule match-ex-table-e-sim*)
apply *simp*
done

from *succsXpt-rec* **obtain** a
where *succsXpt-rec'*:

$a \in set$ (*callers* Π (*Ca*, *Ma*, *pca*))
$\wedge$ (p',*B*) $\in$ *set* (*succsXpt* (Π, *X*, *a* # (*Ca*, *Ma*, *pca*) # *list*))
apply *blast*
done

from *length-L-n* *L-cons* *domC-L* **have** *length-a-L* :*length* (*domC* Π) $-$ *length* (*a* # (*Ca*, *Ma*, *pca*) # *list*) $<$ *n*
by (*simp*, *arith*)

from *succsXpt-rec'* *succsX-domC* *L-cons* *domC-L* *None*
have *succsX-a-domC*: *fst* ` *set* (*succsXpt* (Π, *X*, *a* # (*Ca*, *Ma*, *pca*) # *list*)) $\subset$ *set* (*domC* Π)
apply *simp*
apply (*erule conjE*)+
apply (*simp only*: *in-set-conv-decomp*)
apply (*erule exE*)+
apply (*simp add*: *Pi-def image-Un*)
apply (*simp only*: *insert-def*)
apply (*drule un-subset-drop'*, *assumption*)
apply (*drule match-ex-table-e-sim*)
apply *simp*
done

from *L-cons* **have** *appL-L'-a*: ($\exists L'$. *a* # ((*Ca*, *Ma*, *pca*) # *list*) = L' @ [*p*])
apply (*cut-tac appL-L'*)
apply (*erule exE*)
apply (*rule-tac x=a* # L' **in** *exI*)
apply *simp*
done

from *succsX-a-domC* *succsXpt-rec'* *length-a-L* *IH* *length-L-n* *succsX-domC* *L-cons* *None* *p'-B-succsX-L* *match-ex-tab-L* *callchain-L*
show *?thesis*
apply *simp*
apply (*erule-tac x=length* (*domC* Π) $-$ *length* (*a* # (*Ca*, *Ma*, *pca*) # *list*) **in** *allE*)
apply *simp*
apply (*erule-tac x=a* # (*Ca*, *Ma*, *pca*) # *list* **in** *allE*)
apply *simp*
apply (*cut-tac appL-L'-a*)
apply (*drule mp*, *assumption*)
apply *simp*
done

next

case (*Some aa*)

from *L-cons appL-L′ callchain-L* **have** *domC-Ca-Ma-pca*:(*Ca*,*Ma*,*pca*) ∈ *set* (*domC* Π)
proof (*cases list*)
  case *Nil*
  from *Nil L-cons appL-L′*
  show *?thesis*
    by (*simp add*: *domC-p*)

next
  case *Cons*
  from *Cons callchain-L L-cons appL-L′* **show** *?thesis*
    by (*simp add*: *callers-def*)
qed
        from *Some wf-Pi p-def domC-Ca-Ma-pca Pi-def*
        have *snd-aa-0*:*snd aa* = *0*
          apply (*rule-tac d*=*snd aa* **and** *P*=*P* **and** *An*=*An* **and** *C*=*Ca* **and** *M*=*Ma*
            and *h*=*fst aa* **and** *p*=*pca* **and** *X*=*X* **in** *wf-match-ex-table-d*)
          apply (*simp add*: *wf-def Pi-def*)
          apply (*rule-tac pc*=*pca* **and** *An*=*An* **in** *domC-methodnames*, *simp*)
          apply *simp*
          done
        from *appL-L′* **have** *last-L-p*: *last L* = *p*
          by *fastsimp*

        from *p′-B-succsX-L succsX-domC appL-L′ Pi-def*
        have *B′-xcpt-cond*: ∃ *As*. *B* = *And* (*As*@[*xcpt-cond* (*P*,*An*) *X p*])
          apply −
          apply (*rule-tac L*=*L* **and** *p′*=*p′* **in** *succsXpt-xcpt-cond*)
          apply *simp*
          apply *simp*
          apply *simp*
          done

        from *p′-B-succsX-L valid-B B′-xcpt-cond*
        have *valid-xcpt-cond*: Π,*s* ⊨ (*xcpt-cond* Π *X p*)
          apply −
          apply (*erule exE*)+
          apply (*simp add*: *Pi-def del*: *succsXpt.simps*)
          apply (*simp add*: *evalEs-map*)
          done

        show *?thesis*
        proof (*cases list rule*:*rev-cases*[*consumes 1*,*case-names rev-Nil rev-cons*])
          case *rev-Nil*
          from *rev-Nil appL-L′ L-cons p-def* **have** *p-def′*: (*Ca*,*Ma*,*pca*) = (*C*,*M*,*pc*)

```
    by simp

  from Some rev-Nil p-def' p-def snd-aa-0 X-def
  have find-handler-p:find-handler P xa h ((st,rg,p)#frs) =
    (None,h,([Addr xa],rg,(C,M,fst aa))#frs)
    apply -
    apply simp
    done

  from Some obtain en
    where en-intro: match-ex-table-e P X pca (ex-table-of P Ca Ma) = ⌊en⌋
    and en-def: snd (snd (snd en)) = aa
    apply -
    apply (drule match-ex-table-e-sim2)
    apply (erule exE | erule conjE)+
    apply fastsimp
    done

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


  from p-def p-def' rev-Nil L-cons p'-B-succsX-L en-def en-intro Pi-def succsX-domC
  have p'-def: p' = (C,M,fst aa)
    apply -
    apply simp
    apply (case-tac length (domC (P, An)) ≤ Suc 0)
    apply simp

    apply (simp add: split-def split del: option.split-asm)
    done

    from find-handler-p p'-def show ?thesis
      by fastsimp

next
  case (rev-cons ys y)
  from rev-cons L-cons have L-cons': L = (Ca,Ma,pca)#ys@[p]
    apply simp
    apply (cut-tac appL-L')
    apply (erule exE)
    apply simp
    done
```

from *L-cons′ match-ex-tab-L p-def*
have *no-match-p*: *JVMExceptions.match-ex-table P X pc* (*ex-table-of P C M*) = *None*
by *simp*

from *Some* **obtain** *en*
where *en-intro*: *match-ex-table-e P X pca* (*ex-table-of P Ca Ma*) = ⌊*en*⌋
and *en-def*: *snd* (*snd* (*snd en*)) = *aa*
apply −
apply (*drule match-ex-table-e-sim2*)
apply (*erule exE* | *erule conjE*)+
apply *fastsimp*
done

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

from *en-def en-intro domC-L L-cons′ p′-B-succsX-L*
have *p′-B-def*:*p′* = (*Ca*, *Ma*, *fst aa*) ∧
B = *And* [*Catch X′* (*aF* (*P*, *An*) (*Ca*, *Ma*, *pca*)), *Catch X* (*Pos* (*Ca*, *Ma*, *pca*)),
xcpt-cond (*P*, *An*) *X p*]
apply −
apply (*simp only*: *Pi-def*)
apply (*drule-tac P*=λ *S*. (*p′*, *B*) ∈ (*set S*) **in**
subst[*OF succsXpt-simp*[*of P An X* (*Ca*,*Ma*,*pca*)#(*ys*@[*p*])]])
apply (*simp del*: *succsXpt.simps*)
apply (*simp add*: *split-def*)
done

from *Pi-def s-def* **obtain** *cp ch cst crg cfrs ce*
where *catchstate-s*: *catchstate* (*P*, *X*, *s*) = (*cp*, (*None*, *ch*, (*cst*, *crg*, *cp*) # *cfrs*), *ce*)
apply −
apply (*subgoal-tac* ∃ *p′ h′ st′ rg′ frs′ e′*. *catchstate* (*P*, *X*, *s*) = (*p′*, (*None*, *h′*, (*st′*, *rg′*, *p′*) # *frs′*), *e′*))
prefer *2*
apply (*simp only*:)
apply (*rule catchstate-form′*)
apply (*erule exE*)+
apply *blast*
done

from *Pi-def catchstate-s p′-B-def valid-B s-def*
have *cp-simp*: *cp* = (*Ca*,*Ma*,*pca*)
apply (*simp del*: *catchstate.simps*)
apply (*case-tac frs*)
apply *simp*

apply *simp*
done

from *callers-sysinv-s s-def p-def Pi-def*
have *frs-domC*: $\forall$ *p*$\in$*set* (*map* (*snd* $\circ$ *snd*) ((*st*, *rg*, *C*, *M*, *pc*) # *frs*)). *p* $\in$ *set* (*domC* (*P*, *An*))
apply −
apply (*simp add*: *callers-sysinv-trans del*: *callers-sysinv.simps*)
apply (*case-tac frs*)
apply (*simp add*: *mem-iff*)

apply (*erule conjE*)+
apply (*rule ballI*)
apply (*subgoal-tac* $\exists$ *frs1 frs2*. *frs* = *frs1* @ *pa*#*frs2*)
prefer *2*
apply (*simp only*: *in-set-conv-decomp*)
apply (*erule exE*)+
apply (*erule-tac x*=*length frs1* **in** *allE*)
apply (*drule-tac t*=*a* # *list* **in** *sym*)
apply (*simp add*: *nth-append*)
done

from *cp-simp catchstate-s domC-Ca-Ma-pca s-def frs-domC wf-Pi Pi-def no-match-p Some p-def X-def*
have *find-handler-s*: *find-handler P xa h* ((*st*,*rg*,(*C*,*M*,*pc*))#*frs*) =
(*None*,*h*,([*Addr xa*],*crg*,(*Ca*,*Ma*,*fst aa*))#*cfrs*)
apply −
apply (*rule-tac P*=*P* **and** *C*=*C* **and** *M*=*M* **and** *xa*=*xa* **and** *h*=*h* **and** *e*=*e* **and** *d*=*snd aa*
and *stk*=*st* **and** *rg*=*rg* **and** *rg′*=*crg* **and** *C′*=*Ca* **and** *M′*=*Ma* **and** *pch*=*fst aa* **and** *x′*=*None* **and** *frs′*=*cfrs*
and *An*=*An* **and** *st*=*st* **and** *frs*=*frs* **and** *pc′*=*snd* (*snd cp*) **and** *st′*=*cst* **and** *h′*=*ch* **and** *e′*=*ce*
in *catchstate-find-handler*)
apply (*simp add*: *wf-def*)

apply *assumption*

apply *simp*

```
          apply simp

          apply simp

          apply simp
          done

        from p-def rev-cons L-cons p'-B-succsX-L en-def en-intro Pi-def succsX-domC
        have p'-def: p' = (Ca,Ma,fst aa)
          apply -
          apply simp
          apply (case-tac length (domC (P, An)) ≤ Suc 0)
          apply simp

          apply (case-tac length (domC (P, An)) ≤ Suc (Suc (length ys)))
          apply simp

          apply (simp add: split-def split del: option.split-asm)
          done

        from find-handler-s p'-def p-def
        show ?thesis
          by fastsimp
      qed
    qed
  qed
 qed
qed

from succsXpt-induct[where k=length (domC Π) − 1 and L=[p]]
    succsXpt-domC p'-B-succsXpt Pi-def
show ?thesis
  by (simp del: find-handler.simps)
qed


lemma succsF-progress-Throw:
  assumes Pi-def:Π = (P,An)
  assumes p-def:p = (C,M,pc)
  assumes s-def:s = (p, (None, h, (st, rg, p) # frs), e)
  assumes wf-Pi:wf Π

  assumes p'-B-succsF:(p',B) ∈ set (succsF Π p)
  assumes valid-B:Π,s ⊨ B
```

assumes *cmd-p*:*cmd* Π p = *Some Throw*
assumes *s-inv-Pos*: Π,s $\models$ *inv-Pos* Π (*fst s*)
assumes *s-inv-Ty*:Π,s $\models$ *inv-Ty* Π (*fst s*)
assumes *s-inv-FrNr*:Π,s $\models$ *inv-FrNr* Π (*fst s*)
assumes *s-inv-ExTys*:Π,s $\models$ *inv-ExTys* Π (*fst s*)
shows *goal*: $\exists\, st'\ rg'\ frs'\ e'.\ (s,\ (p',\ (st',\ rg',\ frs'),\ e')) \in$ *effS* Π