# 1 Jinja System Invariants

**theory** $JBC\text{-}SysInv = JBC\text{-}VCG$:

## 1.1 Properties of wellformed programs

**constdefs**
$wf\text{-}md\text{-}Ty{::}jvm\text{-}method\ wf\text{-}mdecl\text{-}test$
$wf\text{-}md\text{-}Ty \ \equiv (\lambda P\ C\ (M,\ Ts,\ T_r,\ mxs,\ mxl_0,\ is,\ xt).\ wt\text{-}method\ P\ C\ Ts\ T_r\ mxs\ mxl_0\ is\ xt\ (pTy\ P\ C\ M))$

**lemma** $wf\text{-}wfprog$:
$wf\ \Pi \Longrightarrow wf\text{-}prog\ wf\text{-}md\text{-}Ty\ (fst\ \Pi)$

**lemma** $wf\text{-}ipc\text{-}domC$:
$wf\ \Pi \Longrightarrow ipc\ \Pi \in set\ (domC\ \Pi)$

**lemma** $main\text{-}method\text{-}no\text{-}args$:
$wf\ (P,An) \Longrightarrow fst\ (snd\ (method\ P\ (fst\ (ipc\ (P,An)))\ (fst\ (snd\ (ipc\ (P,An)))))) = []$

**lemma** $wf\text{-}TypeSafe$:
$wf\ \Pi \Longrightarrow wf\text{-}jvm\text{-}prog\text{-}phi\ (pTy\ (fst\ \Pi))\ (fst\ \Pi)$

**lemma** $wf\text{-}no\text{-}main\text{-}call$:
$wf\ \Pi \Longrightarrow callers\ \Pi\ (C,fst\ (snd\ (ipc\ \Pi)),pc) = []$

**lemma** $handlesEx'\text{-}domC$:
$[\![\ checkExTables\ (P,An);\ p \notin set\ (domC\ (P,An))\ ]\!] \Longrightarrow handlesEx'\ P\ p = []$

**lemma** $methodnames\text{-}split$:
$methodnames\ (P@P') = methodnames\ P\ @\ methodnames\ P'$

**lemma** $methodnames\text{-}split'$:
$methodnames\ (p\#P) = methodnames\ [p]\ @\ methodnames\ P$

**lemma** $domC\text{-}methodnames$:
$(C,M,pc) \in set\ (domC\ (P,An)) \Longrightarrow (C,M) \in set\ (methodnames\ P)$

**constdefs** $get\text{-}mdecl{::}jvm\text{-}prog \Rightarrow cname \Rightarrow mname \Rightarrow mname \times ty\ list \times ty \times jvm\text{-}method$
$get\text{-}mdecl\ P\ C\ M \equiv hd\ (concat\ (map\ (\lambda\ (C',S,Fs,Ms).\ [(M',Ts,T,bs){\in}Ms.\ M = M' \wedge C = C'])\ P))$

**lemma** $get\text{-}mdecl\text{-}append$:
$(C,M) \notin set\ (methodnames\ P) \Longrightarrow get\text{-}mdecl\ (P@P')\ C\ M = get\text{-}mdecl\ P'\ C\ M$

**lemma** *get-mdecl-append'*:
$M \notin set\ (map\ fst\ Ms) \Longrightarrow get\text{-}mdecl\ ((C,S,Fs,Ms@Ms')\#P)\ C\ M = get\text{-}mdecl\ ((C,S,Fs,Ms')\#P)\ C\ M$

**lemma** *methodnames-P*:
$(C,M) \in set\ (methodnames\ P) \Longrightarrow \exists\ ps\ ps'\ S\ Fs\ Ms\ Ms'\ Ts\ T\ bd.\ P = ps@(C,S,Fs,Ms@(M,Ts,T,bd)\#Ms')\#ps' \wedge (C,M) \notin set\ (methodnames\ ps) \wedge M \notin set\ (map\ fst\ Ms) \wedge get\text{-}mdecl\ P\ C\ M = (M,Ts,T,bd)$

**lemma** *methodnames-P2*:
$[\![\ distinct\ (classnames\ P);\ (C,M) \in set\ (methodnames\ P)\ ]\!] \Longrightarrow \exists\ ps\ ps'\ S\ Fs\ Ms\ Ms'\ Ts\ T\ bd.\ P = ps@(C,S,Fs,Ms@(M,Ts,T,bd)\#Ms')\#ps' \wedge (C,M) \notin set\ (methodnames\ ps) \wedge M \notin set\ (map\ fst\ Ms) \wedge get\text{-}mdecl\ P\ C\ M = (M,Ts,T,bd) \wedge class\ P\ C = Some\ (S,Fs,Ms@(M,Ts,T,bd)\#Ms')$

**constdefs** $method'::jvm\text{-}prog \Rightarrow cname \Rightarrow mname \Rightarrow cname \times ty\ list \times ty \times jvm\text{-}method$
$method'\ P\ C\ M \equiv (let\ (M',Ts,T,bd) = get\text{-}mdecl\ P\ C\ M\ in\ (C,Ts,T,bd))$

— method' is defined recursively and is thus easier to evaluate for the simplifier than method, which is defined inductively.

**lemma** *method-method'*:
$[\![\ wf\text{-}prog\ wf\text{-}md\ P;\ (C,M) \in set\ (methodnames\ P)\ ]\!] \Longrightarrow method\ P\ C\ M = method'\ P\ C\ M$

**constdefs** $ex\text{-}table\text{-}of' :: jvm\text{-}prog \Rightarrow cname \Rightarrow mname \Rightarrow ex\text{-}table$
$ex\text{-}table\text{-}of'\ P\ C\ M \equiv snd\ (snd\ (snd\ (snd\ (snd(method'\ P\ C\ M)))))$

**lemma** *ex-table-of-ex-table-of'*:
$[\![\ wf\text{-}prog\ wf\text{-}md\ P;\ (C,M) \in set\ (methodnames\ P)\ ]\!] \Longrightarrow ex\text{-}table\text{-}of\ P\ C\ M = ex\text{-}table\text{-}of'\ P\ C\ M$

**lemma** *match-ex-table-split'*:
$JVMExceptions.match\text{-}ex\text{-}table\ P\ cn\ pc\ et = \lfloor(pc',d)\rfloor \Longrightarrow \exists\ b\ e\ c.\ (b,e,c,pc',d) \in set\ et \wedge P \vdash cn \preceq^*\ c$

**lemma** *match-ex-table-split*:
$match\text{-}ex\text{-}table\ P\ cn\ pc\ et = \lfloor pc'\rfloor \Longrightarrow \exists\ b\ e\ c\ d.\ (b,e,c,pc',d) \in set\ et \wedge P \vdash cn \preceq^*\ c$

**lemma** *wf-ex-table-domC*:
$[\![\ wf\ (P,An);\ (C,M) \in set\ (methodnames\ P);\ match\text{-}ex\text{-}table\ P\ X\ pc\ (ex\text{-}table\text{-}of\ P\ C\ M) = Some\ pc'\ ]\!] \Longrightarrow (C,M,pc') \in set\ (domC\ (P,An))$

**lemma** *handlesEx'-ex-table-of*:
$[\![\ wf\ (P,An);\ (C,M) \in set\ (methodnames\ P)\ ]\!] \Longrightarrow handlesEx'\ P\ (C,M,pc) = remdups'\ (concat\ (map\ (\lambda(b,\ e,\ cn,\ h,\ d).\ if\ pc = h\ then\ [cn]\ else\ [])\ (ex\text{-}table\text{-}of\ P\ C\ M)))$

**lemma** *has-method-class*:
$[\![\ wf\ (P,An);\ has\text{-}method\ P\ C\ M\ ]\!] \Longrightarrow \exists\ Ts\ T\ m.\ method\ P\ C\ M = (C,Ts,T,m)$

**lemma** *wf-extable*:
$[\![$ *wf* $(P,An)$; $(C,M) \in$ *set* (*methodnames* $P$); $(f,t,c,h,d) \in$ *set* (*ex-table-of* $P$ $C$ $M$) $]\!] \Longrightarrow d = 0$

**lemma** *wf-handlesEx′-length*:
*wf* $(P,An) \Longrightarrow$ *length* (*handlesEx′* $P$ $p$) $\leq 1$

**lemma** *wf-domC-cmd*:
**assumes** *wf-Pi*: *wf* $\Pi$
**shows** *set* (*domC* $\Pi$) = $\{p.$ *cmd* $\Pi$ $p \neq$ *None*$\}$
**lemma** *has-method-has-method*:
*TypeRel.has-method* $P$ $C$ $M \Longrightarrow \exists$ $D$ *Ts* $T$ $m$. *has-method* $P$ $D$ $M$ $\wedge$ $P \vdash C \preceq^* D$ $\wedge$ *method* $P$ $C$ $M$ $= (D,Ts,T,m)$ $\wedge$ *method* $P$ $D$ $M$ $= (D,Ts,T,m)$
**lemma** *domC-cmd-instr-of*:
$[\![$ *wf* $\Pi$; $(C,M,pc) \in$ *set* (*domC* $\Pi$) $]\!] \Longrightarrow$ *cmd* $\Pi$ $(C,M,pc)$ = *Some* (*instrs-of* (*fst* $\Pi$) $C$ $M$ ! $pc$)

**lemma** *startstate-initS*:
*wf* $(P,An) \Longrightarrow (p,\sigma,e) \in$ (*initS* $(P,An)) \Longrightarrow \exists$ $C$ $M$ $T$ $m$. $\sigma$ = *start-state* $P$ $C$ $M$ $\wedge$ ($P \vdash C$ *sees* $M$:[]$\rightarrow T$ = $m$ *in* $C$)

**lemma** *wf-safeP-domC*:
$[\![$ *wf* $\Pi$; $(p,\sigma,e) \in$ *safeP* $\Pi$ $]\!] \Longrightarrow p \in$ *set* (*domC* $\Pi$)

**lemma** *drop-length-eq*:
$\bigwedge$ *st st′*. $[\![$ *drop* $k$ *st* = []; *length st* = *length st′* $]\!] \Longrightarrow$ *drop* $k$ *st′* = []

**lemma** *drop-le*: $\bigwedge$ $L$. *drop* $k$ $L$ = [] $\Longrightarrow$ *length* $L \leq k$

**lemma** *jbc-sim-jvm*:
*wf* $(P,An) \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in$ *effS* $(P,An) \Longrightarrow (\sigma,\sigma') \in$ *exec-1* $P$

**lemma** *jbc-sim-jvm-d*:
*wf* $(P,An) \Longrightarrow$ *check* $P$ $\sigma \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in$ *effS* $(P,An) \Longrightarrow P \vdash$ *Normal* $\sigma$ $-jvmd\rightarrow_1$ *Normal* $\sigma'$

**lemma** *effS-jvmd-hull*:
*wf* $(P,An) \Longrightarrow (p,\sigma,e) \in$ *initS* $(P,An) \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in$ (*effS* $(P,An))^* \Longrightarrow P \vdash$ *Normal* $\sigma$ $-jvmd\rightarrow$ *Normal* $\sigma'$

**lemma** *effS-jvm-hull*:
*wf* $(P,An) \Longrightarrow (p,\sigma,e) \in$ *initS* $(P,An) \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in$ (*effS* $(P,An))^* \Longrightarrow P \vdash \sigma$ $-jvm\rightarrow$ $\sigma'$

**lemma** *wf-ipc-no-Invoke*:
**assumes** *wf-Pi*: *wf* $\Pi$
**shows** $\forall$ *Mn* $n$. *cmd* $\Pi$ (*ipc* $\Pi$) $\neq$ *Some* (*Invoke Mn* $n$)

## 1.2 Welltypedness guarantees successful instruction checks.

— checkinstr' is a weaker variant of checkinstr

**consts**
$\mathit{check\text{-}instr}' :: [\mathit{instr}, \mathit{jvm\text{-}prog}, \mathit{heap}, \mathit{val\ list}, \mathit{val\ list}, \mathit{cname}, \mathit{mname}, \mathit{pc}, \mathit{frame\ list}] \Rightarrow \mathit{bool}$

**primrec**

$\mathit{check\text{-}instr}'\text{-}\mathit{Load}$:
$\mathit{check\text{-}instr}'\ (\mathit{Load}\ n)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(n < \mathit{length}\ \mathit{loc})$

$\mathit{check\text{-}instr}'\text{-}\mathit{Store}$:
$\mathit{check\text{-}instr}'\ (\mathit{Store}\ n)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(0 < \mathit{length}\ \mathit{stk} \wedge n < \mathit{length}\ \mathit{loc})$

$\mathit{check\text{-}instr}'\text{-}\mathit{Push}$:
$\mathit{check\text{-}instr}'\ (\mathit{Push}\ v)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(\neg \mathit{is\text{-}Addr}\ v)$

$\mathit{check\text{-}instr}'\text{-}\mathit{New}$:
$\mathit{check\text{-}instr}'\ (\mathit{New}\ C)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$\mathit{is\text{-}class}\ P\ C$

$\mathit{check\text{-}instr}'\text{-}\mathit{Getfield}$:
$\mathit{check\text{-}instr}'\ (\mathit{Getfield}\ F\ C)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(0 < \mathit{length}\ \mathit{stk} \wedge \mathit{is\text{-}Ref}'\ h\ (\mathit{hd}\ \mathit{stk}))$

$\mathit{check\text{-}instr}'\text{-}\mathit{Putfield}$:
$\mathit{check\text{-}instr}'\ (\mathit{Putfield}\ F\ C)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(1 < \mathit{length}\ \mathit{stk} \wedge \mathit{is\text{-}Ref}'\ h\ (\mathit{hd}\ (\mathit{tl}\ \mathit{stk})))$

$\mathit{check\text{-}instr}'\text{-}\mathit{Checkcast}$:
$\mathit{check\text{-}instr}'\ (\mathit{Checkcast}\ C)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(0 < \mathit{length}\ \mathit{stk} \wedge \mathit{is\text{-}class}\ P\ C \wedge \mathit{is\text{-}Ref}'\ h\ (\mathit{hd}\ \mathit{stk}))$

$\mathit{check\text{-}instr}'\text{-}\mathit{Invoke}$:
$\mathit{check\text{-}instr}'\ (\mathit{Invoke}\ M\ n)\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(n < \mathit{length}\ \mathit{stk} \wedge \mathit{is\text{-}Ref}'\ h\ (\mathit{stk}!n) \wedge (\mathit{stk}!n \neq \mathit{Null} \longrightarrow \mathit{TypeRel.has\text{-}method}\ P\ (\mathit{cname\text{-}of}\ h\ (\mathit{the\text{-}Addr}\ (\mathit{stk}!n)))\ \ M))$

$\mathit{check\text{-}instr}'\text{-}\mathit{Return}$:
$\mathit{check\text{-}instr}'\ \mathit{Return}\ P\ h\ \mathit{stk}\ \mathit{loc}\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} =$
$(0 < \mathit{length}\ \mathit{stk})$

*check-instr′-Pop*:
$check\text{-}instr'\ Pop\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(0 < length\ stk)$

*check-instr′-IBin*:
$check\text{-}instr'\ (IBin\ no)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(1 < length\ stk \wedge is\text{-}Intg\ (hd\ stk) \wedge is\text{-}Intg\ (hd\ (tl\ stk)))$

*check-instr′-IfIntCmp*:
$check\text{-}instr'\ (IfIntCmp\ ro\ b)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(1 < length\ stk \wedge is\text{-}Intg\ (hd\ stk) \wedge is\text{-}Intg\ (hd\ (tl\ stk)) \wedge 0 \leq int\ pc + b)$

*check-instr′-IfFalse*:
$check\text{-}instr'\ (IfFalse\ b)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(0 < length\ stk \wedge is\text{-}Bool\ (hd\ stk) \wedge 0 \leq int\ pc{+}b)$

*check-instr′-CmpEq*:
$check\text{-}instr'\ CmpEq\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(1 < length\ stk)$

*check-instr′-Goto*:
$check\text{-}instr'\ (Goto\ b)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(0 \leq int\ pc{+}b)$

*check-instr′-Throw*:
$check\text{-}instr'\ Throw\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs =$
$(0 < length\ stk \wedge is\text{-}Ref'\ h\ (hd\ stk))$

**lemma** *wf-invTys-check′*:
**assumes** *wf-Pi*: $wf\ \Pi$
**assumes** *s-def*: $s = (p,\sigma,e)$
**assumes** *p-def*: $p = (C,M,pc)$
**assumes** *sigma-def*: $\sigma = (None,h,(stk,loc,(C,M,pc))\#frs)$
**assumes** *s-invTy*: $\Pi,s \models inv\text{-}Ty\ \Pi\ p$
**assumes** *cmd-p*: $cmd\ \Pi\ p = Some\ i$
**shows** $check\text{-}instr'\ i\ (fst\ \Pi)\ h\ stk\ loc\ C\ M\ pc\ frs$
**lemma** *Reachables-conv*:
$s \in (Reachables\ \Pi) = (\exists s0.\ s0 \in initS\ \Pi \wedge (s0,\ s) \in (effS\ \Pi)^*\ )$

**lemma** *ReachableFromInv-S-I*:
$s \in ReachableFromInv\ R\ S\ I \Longrightarrow s \in S \vee s \in\ I$

**lemma** *ReachablesAn-ReachableFromInv*:
$\mathit{ReachablesAn}\ \Pi = \mathit{ReachableFromInv}\ (\mathit{effS}\ \Pi)\ (\mathit{initS}\ \Pi)\ \{s.\ \Pi,s \models \mathit{aF}\ \Pi\ (\mathit{fst}\ s)\}$

**lemma** *wf-Reachables-check*:
$[\![\ \mathit{wf}\ \Pi;\ s \in \mathit{Reachables}\ \Pi\ ]\!] \Longrightarrow \mathit{check}\ (\mathit{fst}\ \Pi)\ (\mathit{fst}\ (\mathit{snd}\ s))$

## 1.3 System Exceptions

**lemma** *sys-xcpt-invariant*:
$\bigwedge\ s\ p\ x\ h\ \mathit{frs}\ e.\ [\![\ s{=}(p,(x,h,\mathit{frs}),e);\ s \in \mathit{safeP}\ \Pi;\ C \in \mathit{sys\text{-}xcpts}\ ]\!]$
$\Longrightarrow (\exists\, ob.\ (h\ (\mathit{addr\text{-}of\text{-}sys\text{-}xcpt}\ C)) = \mathit{Some}\ ob \wedge \mathit{obj\text{-}ty}\ ob = \mathit{Class}\ C)$

**lemma** *sys-xcpt-class*:
$[\![\ \mathit{wf}\ \Pi;\ C \in \mathit{sys\text{-}xcpts}\ ]\!] \Longrightarrow (\exists\ \mathit{fs}\ \mathit{ms}.\ \mathit{class}\ (\mathit{fst}\ \Pi)\ C = \mathit{Some}\ (\mathit{Exception},\mathit{fs},\mathit{ms}))$

**lemma** *callstates-callers-sysinv*:
$\bigwedge\ s\ s'.\ [\![\ \mathit{callers\text{-}sysinv}\ (\Pi,s);\ s' \in (\mathit{callstates}\ s)\ ]\!] \Longrightarrow \mathit{callers\text{-}sysinv}\ (\Pi,s')$

**lemma** *Reachables-pos*:
$[\![\ \mathit{wf}\ \Pi;\ (p,(x,h,(\mathit{st},\mathit{rg},p')\#\mathit{frs}),e) \in \mathit{Reachables}\ \Pi\ ]\!] \Longrightarrow p = p'$

**lemma** *safeP-pos*:
$[\![\ \mathit{wf}\ \Pi;\ (p,(x,h,(\mathit{st},\mathit{rg},p')\#\mathit{frs}),e) \in \mathit{safeP}\ \Pi\ ]\!] \Longrightarrow p = p'$

**lemma** *callers-sysinv-domC*:
$\bigwedge\ p\ x\ h\ e.\ \mathit{callers\text{-}sysinv}\ (\Pi,(p,(x,h,\mathit{frs}),e)) \Longrightarrow\ \forall\ p \in \mathit{set}\ (\mathit{map}\ (\mathit{snd} \circ \mathit{snd})\ \mathit{frs}).\ p \in \mathit{set}\ (\mathit{domC}\ \Pi)$

**lemma** *catchstate-eq*:
$\forall\ P\ X\ p\ x\ h\ h'\ \mathit{fr}\ \mathit{fr}'\ \mathit{fc}\ \mathit{frs}\ e.\ \mathit{catchstate}\ (P,X,(p,(x,h,\mathit{fr}\#\mathit{fc}\#\mathit{frs}),e)) = \mathit{catchstate}\ (P,X,(p,(x,h',\mathit{fr}'\#\mathit{fc}\#\mathit{frs}),e))$

**lemma** *callers-eq*:
$\mathit{callers}\ \Pi\ (C,M,\mathit{pc}) = \mathit{callers}\ \Pi\ (C,M,\mathit{pc}')$

**lemma** *callers-eq-Suc*: $\mathit{callers}\ \Pi\ (C,M,\mathit{Suc}\ \mathit{pc}) = \mathit{callers}\ \Pi\ (C,M,\mathit{pc})$

**lemma** *callers-eq-Add*: $\mathit{callers}\ \Pi\ (C,M,\mathit{nat}\ (\mathit{int}\ \mathit{pc}) + b) = \mathit{callers}\ \Pi\ (C,M,\mathit{pc})$

**lemma** *callers-eq-Add2*: $\mathit{callers}\ \Pi\ (C,M,\mathit{nat}\ (\mathit{int}\ \mathit{pc} + b)) = \mathit{callers}\ \Pi\ (C,M,\mathit{pc})$

**lemma** *callers-sysinv-env*:
$\bigwedge\ p\ h\ h'\ e\ e'.\ \mathit{callers\text{-}sysinv}\ (\Pi,(p,(\mathit{None},h,\mathit{frs}),e)) = \mathit{callers\text{-}sysinv}\ (\Pi,(p,(\mathit{None},h',\mathit{frs}),e'))$

**lemma** *match-ex-table-handlesEx'*:
$[\![\ \mathit{wf}\ (P,\mathit{An});\ (C,M) \in \mathit{set}\ (\mathit{methodnames}\ P);\ \mathit{match\text{-}ex\text{-}table}\ P\ \mathit{cn}\ \mathit{pc}\ (\mathit{ex\text{-}table\text{-}of}\ P\ C\ M) = \mathit{Some}\ \mathit{pc}'$
$]\!] \Longrightarrow \exists\,\mathit{cns}\ \mathit{cn}'\ \mathit{cns}'.\ \mathit{handlesEx}'\ P\ (C,M,\mathit{pc}') = \mathit{cns}@\ \mathit{cn}'\#\mathit{cns}' \wedge\ P \vdash \mathit{cn} \preceq^*\ \mathit{cn}'$

**lemma** *handlesEx-domC*:
$[\![$ *wf* $(P,An)$; *handlesEx P p* $=$ *Some C* $]\!] \Longrightarrow p \in$ *set* (*domC* $(P,An)$)

**lemma** *findhandler-handlesEx′*:
$\bigwedge p'$. $[\![$ *wf* $(P,An)$; *find-handler P xa h frs* $=$ (*None*, *h*, ([*Addr xa*], *loc′*, *p′*) # *frs′*); *handlesEx′ P p′* $\neq$ [] $]\!]$
$\Longrightarrow \exists$ *cns cn cns′*. *handlesEx′ P p′* $=$ *cns@cn# cns′* $\wedge P \vdash$ (*cname-of h xa*) $\preceq^*$ *cn*
**lemma** *exception-ext-object*:
*wf* $(P,An) \Longrightarrow$ (*Exception*,*Object*) $\in$ *subcls1 P*
**apply** (*rule subcls1I*)
**apply** (*simp add*: *wf-def*)
**apply** (*erule conjE* | *erule exE*)+
**apply** (*simp add*: *SystemClasses-def Exception-def Object-def class-def ObjectC-def ExceptionC-def*)

**apply** (*simp add*: *Exception-def Object-def*)
**done**

**lemma** *exception-ext-object-only*:
*wf* $(P,An) \Longrightarrow$ (*Exception*,*c*) $\in$ *subcls1 P* $\Longrightarrow$ *c* $=$ *Object*
**apply** (*erule subcls1.elims*)
**apply** (*simp add*: *wf-def*)
**apply** (*erule conjE* | *erule exE*)+
**apply** (*simp add*: *SystemClasses-def Exception-def Object-def class-def ObjectC-def ExceptionC-def*)
**done**

**lemma** *no-cyclic-inheritance*:
*wf* $(P,An) \Longrightarrow (c,c) \notin$ (*subcls1 P*)
**sorry**

**lemma** *object-sup*:
*wf* $(P,An) \Longrightarrow$ (*Object*,*c*) $\notin$ (*subcls1 P*)
**sorry**

**lemma** *sys-xcpts-ext-exception*:
$[\![$ *wf* $(P,An)$; *cn* $\in$ *sys-xcpts* $]\!] \Longrightarrow$ (*cn*,*Exception*) $\in$ *subcls1 P*
**apply** (*rule subcls1I*)
**apply** (*simp add*: *wf-def*)
**apply** (*erule conjE* | *erule exE*)+
**apply** (*simp add*: *sys-xcpts-def OutOfMemory-def ClassCast-def NullPointer-def*
*SystemClasses-def NullPointerC-def OutOfMemoryC-def ClassCastC-def*
*Exception-def Object-def class-def ObjectC-def ExceptionC-def*)
**apply** (*erule disjE* | *simp*)+

**apply** (*simp add*: *sys-xcpts-def OutOfMemory-def ClassCast-def NullPointer-def*

*SystemClasses-def NullPointerC-def OutOfMemoryC-def ClassCastC-def Exception-def Object-def class-def ObjectC-def ExceptionC-def*)
**apply** (*erule disjE* | *simp*)+
**done**

**lemma** *sysxpct-no-object*:
$\bigwedge cn\ cn'.\ [\![\ wf\ (P,An);\ P \vdash cn' \preceq^* cn;\ cn \notin \{Exception,Object\};\ cn' \in sys\text{-}xcpts\ ]\!] \Longrightarrow cn = cn'$

**lemma** *findhandler-handlesEx*:
$[\![\ wf\ (P,An);\ find\text{-}handler\ P\ xa\ h\ frs = (None,\ h,\ ([Addr\ xa],\ loc',\ p')\ \#\ frs');$
$handlesEx\ P\ p' = Some\ cn\ ]\!] \Longrightarrow P \vdash cname\text{-}of\ h\ xa \preceq^* cn$

**lemma** *match-ex-table-subtype-None*:
$[\![\ match\text{-}ex\text{-}table\ P\ cn\ pc\ et = None;\ \ P \vdash\ \ cn \preceq^* cn'\ ]\!]$
$\Longrightarrow match\text{-}ex\text{-}table\ P\ cn'\ pc\ et = None$

**lemma** *handlesEx-match-ex-table*:
$\bigwedge d.\ [\![\ wf\ (P,An);\ (C,M) \in set\ (methodnames\ P);\ handlesEx\ P\ (C,M,pc') = Some\ cn;$
$match\text{-}ex\text{-}table\ P\ cn'\ pc\ (ex\text{-}table\text{-}of\ P\ C\ M) = Some\ pc'\ ]\!]$
$\Longrightarrow match\text{-}ex\text{-}table\ P\ cn\ pc\ (ex\text{-}table\text{-}of\ P\ C\ M) = Some\ pc'$

**lemma** *find-handler-catchstate*:
$\bigwedge st'\ loc\ p\ e.\ [\![\ wf\ (P,An);\ find\text{-}handler\ P\ xa\ h\ ((st',loc,p)\#frs) =$
$(None,h,([Addr\ xa],loc',C',M',pc')\#frs');\ handlesEx\ P\ (C',M',pc') = Some\ cn\ ]\!]$
$\Longrightarrow (\exists h'\ stk'\ pc''.\ catchstate\ (P,cn,(p,(None,h,(stk,loc,p)\#frs),e)) =$
$((C',M',pc''),(None,h',(stk',loc',C',M',pc'')\#frs'),e(\!|cs{:=}drop\ (length\ frs - length\ frs')\ (cs\ e)|\!))$
$\wedge\ (match\text{-}ex\text{-}table\ P\ (cname\text{-}of\ h\ xa)\ pc''\ (ex\text{-}table\text{-}of\ P\ C'\ M')) = Some\ pc') \vee frs = frs'$

**lemma** *find-handler-catchstate'*:
$\bigwedge st'\ loc\ p\ e.\ [\![\ find\text{-}handler\ P\ xa\ h\ ((st',loc,p)\#frs)=(None,h,([Addr\ xa],loc',C',M',pc')\#frs')$
$]\!] \Longrightarrow (\exists h'\ stk'\ pc''.\ catchstate\ (P,(fst\ (the\ (h\ xa))),(p,(None,h,(stk,loc,p)\#frs),e)) =$
$((C',M',pc''),(None,h',(stk',loc',C',M',pc'')\#frs'),e(\!|cs{:=}drop\ (length\ frs - length\ frs')\ (cs\ e)|\!))$
$\wedge\ (match\text{-}ex\text{-}table\ P\ (fst\ (the\ (h\ xa)))\ pc''\ (ex\text{-}table\text{-}of\ P\ C'\ M')) = Some\ pc') \vee frs = frs'$
**lemma** *callers-sysinv-Reachables*:
$[\![\ wf\ \Pi;\ s \in Reachables\ \Pi\ ]\!] \Longrightarrow callers\text{-}sysinv\ (\Pi,s)$

**lemma** *safeP-callers-sysinv*:
$[\![\ wf\ \Pi;\ s \in safeP\ \Pi\ ]\!] \Longrightarrow callers\text{-}sysinv\ (\Pi,s)$

**lemma** *find-handler-frs'*:
$find\text{-}handler\ P\ x\ h\ frs = (None,h,fr\#frs')$
$\Longrightarrow \exists pfx.\ frs = pfx\ @\ frs' \wedge pfx \neq [] \wedge (let\ (C0,M0,pc0) = snd\ (snd\ fr);$

$(C0',M0',pc0') = (snd\ (snd\ (last\ pfx)))$
$in\ (C0 = C0' \wedge M0 = M0'))$

**lemma** *Reachables-state*:
**assumes** *s-Reach*: $s \in Reachables\ \Pi$
**shows** $\exists\ C\ M\ pc\ h\ st\ rg\ frs\ e.\ s = ((C,M,pc),(None,h,(st,rg,(C,M,pc))\#frs),e)$
$\wedge\ (let\ (C0,M0,pc0) = last\ (map\ (snd \circ snd)\ ((st,rg,(C,M,pc))\#frs));$
$(C0',M0',pc0') = ipc\ \Pi$
$in\ (C0 = C0' \wedge M0 = M0'))$ **using** *s-Reach*

**lemma** *wf-Reachables-domC*:
$[\![\ wf\ \Pi;\ s \in Reachables\ \Pi\ ]\!] \Longrightarrow fst\ s \in set\ (domC\ \Pi)$

**lemma** *wf-Reachables-domC'*:
$\bigwedge\ p\ x\ h\ e.\ [\![\ wf\ \Pi;\ (p,(x,h,frs),e) \in Reachables\ \Pi\ ]\!] \Longrightarrow \forall\ p \in set\ (map\ (snd \circ snd)\ frs).\ p \in set\ (domC\ \Pi)$

**lemma** *wf-safeP-domC'*:
$\bigwedge\ p\ x\ h\ e.\ [\![\ wf\ \Pi;\ (p,(x,h,frs),e) \in safeP\ \Pi\ ]\!] \Longrightarrow \forall\ p \in set\ (map\ (snd \circ snd)\ frs).\ p \in set\ (domC\ \Pi)$

**lemma** *callers-sysinv-trans*:
$\bigwedge\ p\ x\ h\ e.\ callers\text{-}sysinv\ (\Pi,(p,(x,h,frs),e)) = (\forall\ i < length\ frs.\ snd\ (snd\ (frs\ !\ i)) \in set\ (domC\ \Pi)\ \wedge$
$(if\ Suc\ i = length\ frs$
$then\ fst\ (snd\ (snd\ (frs\ !\ i))) = fst\ (ipc\ \Pi)$
$\wedge\ fst\ (snd\ (snd\ (snd\ (frs\ !\ i)))) = fst\ (snd\ (ipc\ \Pi))$
$else\ snd\ (snd\ (frs\ !\ (Suc\ i))) \in set\ (callers\ \Pi\ (snd\ (snd\ (frs\ !\ i))))))$

**lemma** *no-recursive-main*:
$[\![\ wf\ \Pi;\ s \in Reachables\ \Pi\ ]\!] \Longrightarrow$
$fst\ (snd\ (ipc\ \Pi)) \notin set\ (butlast\ (map\ (fst\ o\ snd\ o\ snd\ o\ snd)\ (snd\ (snd\ (fst\ (snd\ s))))))$

## 1.4 Frame Stack Size

**lemma** *inv-FrNr-Reachable*:
$[\![\ wf\ \Pi;\ s \in Reachables\ \Pi\ ]\!] \Longrightarrow \Pi,s \models inv\text{-}FrNr\ \Pi\ (fst\ s)$

## 1.5 Position information

**lemma** *addPos-append*:
$addPos\ p\ (es@es') = addPos\ p\ es\ @\ addPos\ p\ es'$

**lemma** *addPos-cons*:
$addPos\ p\ ((p',B)\#es) = addPos\ p\ [(p',B)]\ @\ addPos\ p\ es$

**lemma** *addPos-single*:

$addPos\ p\ [(p',\ B)] = [(p',\ And\ [Pos\ p,B])]$

**lemma** *addPos-map-fst*:
$map\ fst\ (addPos\ p\ xs) = map\ fst\ xs$

**lemma** *inv-Pos-Reachable*:
$[\![\ wf\ \Pi\ ;\ s \in Reachables\ \Pi\ ]\!] \Longrightarrow \Pi,s \models inv\text{-}Pos\ \Pi\ (fst\ s)$

## 1.6 System Exception Types

**lemma** *inv-ExTys-Reachable*:
$s \in Reachables\ \Pi \Longrightarrow \Pi,\ s \models inv\text{-}ExTys\ \Pi\ (fst\ s)$

**lemma** *extractTy-sem*:
$\bigwedge tys.\ [\![\ \Pi,s \models A;\ extractTy\ (A,ex) = tys;\ tys \neq []\ ]\!] \Longrightarrow \exists\ tp \in set\ tys.\ \Pi,s \models Ty\ ex\ tp$

## 1.7 Type Conformance

**lemma** *methodnames-prog-kil*:
$(C,M) \in set\ (methodnames\ P) \Longrightarrow (\exists\ \tau.\ (prog\text{-}kil\ P)\ ?\ (C,M) = Some\ \tau)$

**lemma** *state-format*: $[\![\ s0 \in initS\ \Pi;\ (s0,s)\in (effS\ \Pi)^*\ ]\!] \Longrightarrow \exists\ p\ h\ st\ rg\ frs\ e.\ s = (p,(None,h,(st,rg,p)\#frs),e)$

**lemma** *inv-Ty-Reachable*:
**assumes** *wf-Pi*: $wf\ \Pi$
**assumes** *s-Reachables*: $s \in Reachables\ \Pi$
**shows** $\Pi,s \models inv\text{-}Ty\ \Pi\ (fst\ s)$

**lemma** *statesplit-Reachables*:
$s \in Reachables\ \Pi \Longrightarrow \exists\ C\ M\ pc\ h\ st\ rg\ frs\ e.\ s = ((C,M,pc),(None,h,(st,rg,(C,M,pc))\#frs),e) \wedge fst\ (snd\ (last\ (map\ (snd \circ snd)\ ((st,rg,(C,M,pc))\#frs))\ )) = fst\ (snd\ (ipc\ \Pi))$

**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** *callers-sysinv-pos*:
$callers\text{-}sysinv\ (\Pi,(p,(x,h,frs),e)) = callers\text{-}sysinv\ (\Pi,(p',(x,h,frs),e))$

**lemma** *callers-sysinv-append*:
$\bigwedge p\ fr\ fr'\ frs'\ h\ e.\ callers\text{-}sysinv\ (\Pi,(p,(None,h,frs@fr\#fr'\#frs')::jvm\text{-}state,e)) \Longrightarrow snd\ (snd\ fr') \in set\ (callers\ \Pi\ (snd\ (snd\ fr))) \wedge (\forall\ p'\ h'\ e'.\ callers\text{-}sysinv\ (\Pi,(p',(None,h',fr'\#frs'),e')))$
**proof** (*induct frs*)

**case** *Nil*
**from** *Nil* **show** *?case*
**apply** −
**apply** (*case-tac frs'::val list ~~> val list* × *char list* × *char list* × *nat*)
**apply** (*simp add*: *split-def*)

**apply** (*simp add*: *split-def*)
**apply** (*rule allI*)+
**apply** (*erule conjE*)+
**apply** (*case-tac list*)
**apply** *simp*

**apply** (*simp add*: *split-def del*: *callers-sysinv.simps*)
**apply** (*subgoal-tac callers-sysinv* (Π, *snd* (*snd aa*), (*None*, *hd* (*tl* (*tl* (*cs e*))), *aa* # *lista*), *e*⦇*cs* := *tl* (*tl* (*tl* (*cs e*)))⦈) =
*callers-sysinv* (Π, *snd* (*snd aa*), (*None*, *hd* (*tl* (*cs e'*)), *aa* # *lista*), *e'*⦇*cs* := *tl* (*tl* (*cs e'*))⦈))
**prefer** *2*
**apply** (*rule callers-sysinv-env*)
**apply** *simp*
**done**

**next**

**case** (*Cons fr'' frs''*)
**from** *Cons* **show** *?case*
**apply** −
**apply** (*drule-tac P*=λ *x*. *x* **in** *subst*[*OF callers-sysinv.simps*])
**apply** (*simp del*: *callers-sysinv.simps*)
**apply** (*case-tac* (*frs''* @ *fr* # *fr'* # *frs'*))
**apply** (*simp del*: *callers-sysinv.simps*)

**apply** (*simp only*: *list.cases*)
**apply** (*drule-tac t*=*a* # *list* **in** *sym*)
**apply** (*simp only*:)
**apply** (*simp add*: *Let-def split-def fst-conv snd-conv del*: *callers-sysinv.simps*)
**apply** *atomize*
**apply** (*erule-tac x*=*snd* (*snd a*) **in** *allE*)
**apply** (*erule-tac x*=*hd* (*cs e*) **in** *allE*)
**apply** (*erule-tac x*=*fr* **in** *allE*)
**apply** (*erule-tac x*=*fr'* **in** *allE*)
**apply** (*erule-tac x*=*frs'* **in** *allE*)
**apply** (*erule-tac x*=*e*⦇*cs* := *tl* (*cs e*)⦈ **in** *allE*)
**apply** (*erule conjE*)+
**apply** (*drule mp*, *assumption*)

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

**lemma** *statesplit-Pos*:
**assumes** *s-Pos*: $\Pi$,$s \models Pos\ (fst\ s)$
**shows** $\exists\ C\ M\ pc\ h\ st\ rg\ frs\ e.\ s = ((C,\ M,\ pc),\ (None,\ h,\ (st,\ rg,\ C,\ M,\ pc)\ \#\ frs),\ e)$

**lemma** *callers-sysinv-Pos*: $\Pi,s \models Pos\ (fst\ s) \Longrightarrow callers\text{-}sysinv\ (\Pi,s)$

**constdefs** $correctAn::jbc\text{-}prog \Rightarrow bool$
$correctAn\ \Pi \equiv (\forall\ s \in Reachables\ \Pi.\ (\forall\ An.\ (anF\ \Pi\ (fst\ s) = Some\ An) \longrightarrow \Pi,s \models An))$

**lemma** *correct-state-inv*:
**assumes** $wt{:}wf\text{-}jvm\text{-}prog_k\ P$
**assumes** $sees\text{-}mthd{:}P \vdash C\ sees\ M{:}[]{\rightarrow}T = m\ in\ C$
**assumes** $reachable{:}P \vdash start\text{-}state\ P\ C\ M\ {-jvm}{\rightarrow}\ \sigma'$
**shows** $\exists\ \Phi.\ P,\Phi \vdash \sigma'\ \surd$

## 1.8 Wellformed Control Flow

**lemma** *succsF-domC*:
$[\![\ wf\ \Pi;\ (p',B) \in set\ (succsF\ \Pi\ p)\ ]\!] \Longrightarrow (p \in set\ (domC\ \Pi) \wedge p' \in set\ (domC\ \Pi))$
**lemma** *succsXpt-xcpt-cond*:
$\bigwedge L.\ [\![\ fst\ `\ set\ (succsXpt\ (\Pi,\ X,\ L)) \subset set\ (domC\ \Pi);$
$(p',B) \in set\ (succsXpt\ (\Pi,\ X,\ L));$
$(\exists\ L'.\ L = L'@[p])\ ]\!]$
$\Longrightarrow \exists\ As.\ B = And\ (As@[xcpt\text{-}cond\ \Pi\ X\ p])$
**lemma** *wf-succsXpt-xcpt-cond*:
**assumes** *wf-Pi*: $wf\ \Pi$
**and** *cmd-p*: $cmd\ \Pi\ p = Some\ i$
**and** *i-not-Throw*: $i \neq Throw$
**and** $p'$*-succsExcept-p*: $(p',B) \in set\ (succsExcept\ \Pi\ p)$
**shows** $\exists\ As.\ B = And\ (As\ @\ [xcpt\text{-}cond\ \Pi\ (sys\text{-}xcpt\text{-}of\ i)\ p])$
**proof** −

**from** *cmd-p*
**have** *p-domC*: $p \in set\ (domC\ \Pi)$
**by** (*rule cmd-domC*)

**from** *p-domC* **obtain** $dC\ dC'$
**where** $dC\text{-}p\text{-}dC'$: $domC\ \Pi = dC\ @\ p\ \#\ dC'$
**apply** −
**apply** (*simp add*: *in-set-conv-decomp*)

**apply** *fastsimp*
**done**

**from** *dC-p-dC′ wf-Pi cmd-p p′-succsExcept-p i-not-Throw*
**have** *subset-domC*:*fst* ‘ *set* (*succsXpt* (Π, (*sys-xcpt-of i*), [*p*])) ⊂ *set* (*domC* Π)
**apply** −
**apply** (*simp add*: *wf-def checkPos-split succsExcept-def*
*split add*: *split-if-asm split del*: *option.split-asm option.split*)
**apply** (*case-tac i*)
**apply** (*simp add*: *sys-xcpt-of-def split del*: *option.split-asm option.split*)+
**done**

**from** *p′-succsExcept-p i-not-Throw cmd-p*
**have** *p′-succsXpt-p*: (*p′*,*B*) ∈ *set* (*succsXpt* (Π,*sys-xcpt-of i*,[*p*]))
**apply** −
**apply** (*simp add*: *succsExcept-def split del*: *option.split-asm option.split*)
**apply** (*case-tac i*)
**apply** (*simp add*: *sys-xcpt-of-def split del*: *option.split-asm option.split*)+
**done**

**from** *p′-succsXpt-p subset-domC* **show** *?thesis*
**apply** −
**apply** (*rule succsXpt-xcpt-cond*)
**apply** *assumption*
**apply** *assumption*
**apply** *simp*
**done**
**qed**

**end**