1 wpF computes weakest preconditions

theory *JBC-wpFcorrect* = *JBC-SysInv*:

1.1 Alternative wpF Defintions

lemma *wpF-Load*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (Load\ n)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ ((map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))@$
$(map\ (\lambda k.\ (St\ k, if\ k{=}0\ then\ Rg\ n$
$else\ St\ (k - 1)))\ (stkIds\ Q)))\ Q$

lemma *wpF-Store*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (Store\ n)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ ((map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))@$
$((Rg\ n, St\ 0)\#\ map\ (\lambda k.\ (St\ k,\ St\ (k{+}1)))\ (stkIds\ Q)))\ Q$

lemma *wpF-Push*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (Push\ v)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ ((map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))@$
$(map\ (\lambda k.\ (St\ k, if\ k{=}0\ then\ Cn\ v\ else\ St\ (k - 1)))\ (stkIds\ Q)))\ Q$

lemma *wpF-New*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (New\ Cl)$
shows $wpF\ \Pi\ p\ p'\ Q = (let\ em{=}((map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))$
$@(map\ (\lambda k.\ (St\ k, if\ k{=}0\ then\ NewA\ 0$
$else\ St\ (k - 1)))\ (stkIds\ Q))@$
$(map\ (\lambda n.\ (NewA\ n,\ NewA\ (n{+}1)))\ (getNewEx\ Q)));$
$gfe'{=}foldl\ (\lambda mp\ hex.\ (case\ hex$
$of\ GF\ F\ C\ ex \Rightarrow (let\ ex'{=}substE\ mp\ ex$
$in\ (Gf\ F\ C\ ex, IF\ ex' \doteq NewA\ 0$
$THEN\ Cn\ (the\ ((snd\ (blank\ (fst\ \Pi)\ Cl))(F,C)))$
$ELSE\ Gf\ F\ C\ ex'))$
$|\ TY\ ex\ ty\ \ \Rightarrow (let\ ex'{=}substE\ mp\ ex$
$in\ (Ty\ ex\ ty, IF\ ex' \doteq NewA\ 0$
$THEN\ Cn\ (Bool\ ((Class\ Cl) = ty))$
$ELSE\ Ty\ ex'\ ty)))\#mp)$
$em\ (remdups'\ (getHeapEx\ Q))$
$in\ substE\ gfe'\ Q)$

lemma *wpF-Getfield*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$

assumes *cmd-p*: *cmd* Π *p* = *Some* (*Getfield F C*)
shows *wpF* Π *p p′ Q* = *substE* ((*map* (λ*q*. (*Pos q*, *if q* = *p′ then Pos p else FF*)) (*getPosEx Q*))@[(*St 0*,*Gf F C* (*St 0*))]) *Q*

lemma *wpF-Putfield*:
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some* (*Putfield F C*)
shows *wpF* Π *p p′ Q* = (*let em*=(*map* (λ*q*. (*Pos q*, *if q* = *p′ then Pos p else FF*)) (*getPosEx Q*))
@(*map* (λ*k*. (*St k*,*St* (*k*+*2*))) (*stkIds Q*));
gfe′=*foldl* (λ*mp ex*. *let ex′*=*substE mp ex*
in (*Gf F C ex*,*IF* (*ex′* ≐ *St 1*)
THEN St 0 ELSE Gf F C ex′)#*mp*)
em (*remdups′* (*getGfEx F C Q*))
in substE gfe′ Q)

lemma *wpF-Checkcast*:
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some* (*Checkcast C*)
shows *wpF* Π *p p′ Q* = *substE* (*map* (λ*q*. (*Pos q*, *if q* = *p′ then Pos p else FF*)) (*getPosEx Q*)) *Q*

lemma *wpF-Invoke*:
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some* (*Invoke M n*)
shows *wpF* Π *p p′ Q* = *substE* ((*map* (λ*q*. (*Pos q*, *if q* = *p′ then Pos p else FF*)) (*getPosEx Q*))@(*FrNr*,*FrNr* ⊕ (*Cn* (*Intg 1*)))#
(*map* (λ*k*. (*Rg k*,*if k* ≤ *n then St* (*n*−*k*)
else (*if k* ≤ *n* + *fst* (*snd* (*snd* (*snd* (*snd* (*method* (*fst* Π) (*fst p′*) *M*)))))
then Cn arb
else none))) (*rgIds Q*))@
(*map* (λ*k*. (*St k*,*none*)) (*stkIds Q*))@
(*map* (λ*ex*. (*Call ex*,*ex*)) (*getCallEx Q*))@
(*concat* (*map* (λ(*cn′*,*ex′*).
(*if catchesEx* (*fst* Π) *cn′ p*
then [(*Catch cn′ ex′*,*ex′*)]
else [(*Catch cn′ ex′*,
IF (*FrNr* ≐ *Cn* (*Intg 1*)) *THEN ex′*
ELSE Catch cn′ ex′)])) (*getCatchEx Q*)))) *Q*

lemma *wpF-Return*:
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some Return*
shows *wpF* Π *p p′ Q* = (*let* (*C*,*M*,*pc*)=*p*; (*P*,*An*)=Π; *n* = *length* (*fst* (*snd* (*method P C M*)))
in substE ((*map* (λ*q*. (*Pos q*, *if q* = *p′ then Pos p else FF*)) (*getPosEx Q*))
@(*FrNr*,*FrNr* ⊖ (*Cn* (*Intg 1*)))#
(*map* (λ*k*. (*St k*,*if 1* ≤ *k then Call* (*St* (*n*+*k*))
else St 0)) (*stkIds Q*))@

$(map\ (\lambda k.\ (Rg\ k, Call\ (Rg\ k)))\ (rgIds\ Q))@$
$(map\ (\lambda ex.\ (Call\ ex, Call\ (Call\ ex)))\ (getCallEx\ Q))@$
$(map\ (\lambda(cn', ex').\ (Catch\ cn'\ ex', Call\ (Catch\ cn'\ ex')))$
$(getCatchEx\ Q)))\ Q)$

lemma *wpF-Pop*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ Pop$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q)$
$@(map\ (\lambda k.\ (St\ k, St\ (k+1)))\ (stkIds\ Q)))\ Q$

lemma *wpF-IBin*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (IBin\ no)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q)$
$@\ (map\ (\lambda k.\ (St\ k, if\ k{=}0\ then\ Num\ (St\ 1)\ no\ (St\ 0)\ else\ (St\ (k+1))))\ (stkIds$
$Q)))\ Q$

lemma *wpF-Goto*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (Goto\ t)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))\ Q$

lemma *wpF-CmpEq*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ CmpEq$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q)$
$@(map\ (\lambda k.\ (St\ k, if\ k{=}0\ then\ (St\ 0) \doteq (St\ 1)$
$else\ (St\ (k+1))))\ (stkIds\ Q)))\ Q$

lemma *wpF-IfIntCmp*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (IfIntCmp\ ro\ t)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q)$
$@(map\ (\lambda k.\ (St\ k, St\ (k+2)))\ (stkIds\ Q)))\ Q$

lemma *wpF-IfFalse*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ (IfFalse\ t)$
shows $wpF\ \Pi\ p\ p'\ Q = substE\ (map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q)$
$@(map\ (\lambda k.\ (St\ k, St\ (k+1)))\ (stkIds\ Q)))\ Q$

lemma *wpF-Throw-Nrm*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ Throw$

shows $wpF\ \Pi\ p\ p'\ Q = FF$

lemma *wpF-Except*:
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = Some\ cn$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
shows $wpF\ \Pi\ p\ p'\ Q = (let\ mp{=}(map\ (\lambda q.\ (Pos\ q,\ if\ q = p'\ then\ Pos\ p\ else\ FF))\ (getPosEx\ Q))\ @$
$(map\ (\lambda k.\ (St\ k, if\ 1{\le}k\ then\ none$
$else\ (if\ (i = Throw)$
$then\ (IF\ St\ 0 \doteq Cn\ (Null)$
$THEN\ (Cn\ (Addr\ (addr\text{-}of\text{-}sys\text{-}xcpt\ NullPointer)))$
$ELSE\ St\ 0)$
$else\ Cn\ (Addr\ (addr\text{-}of\text{-}sys\text{-}xcpt\ (sys\text{-}xcpt\text{-}of\ i))))))\ (stkIds\ Q))@$
$(let\ (C,M,pc){=}p;\ (C',M',pc'){=}p';\ (P,An){=}\Pi\ in$
$(if\ match\text{-}ex\text{-}table\ P\ cn\ pc\ (ex\text{-}table\text{-}of\ P\ C\ M) = Some\ pc'\ then\ []$
$else$
$let\ rgm{=}map\ (\lambda k.\ (Rg\ k, Catch\ cn\ (Rg\ k)))\ (rgIds\ Q);$
$om\ {=}map\ (\lambda ex.\ (Call\ ex, Catch\ cn\ (Call\ ex)))\ (getCallEx\ Q);$
$cm{=}\ map\ (\lambda(cn',ex').\ (Catch\ cn'\ ex',\ Catch\ cn\ (Catch\ cn'\ ex')))$
$(getCatchEx\ Q)$
$in\ (FrNr, Catch\ cn\ FrNr)\#rgm@om@cm))$
$in\ substE\ mp\ Q)$

1.2 Auxiliary Definitions and Lemmas

lemma *foldl-map-lookup'*:
$\bigwedge\ es.\ \forall\, hex \in set\ es.\ heapEx\ x \neq [hex] \implies \forall\, mp'.\ (foldl\ (\lambda mp\ hex.\ (case\ hex$
$of\ GF\ F\ C\ ex \Rightarrow (let\ ex'{=}substE\ mp\ ex\ in\ (Gf\ F\ C\ ex,$
$IF\ ex' \doteq NewA\ 0\ THEN\ Cn\ (the\ (snd\ (blank\ P\ Cl)\ (F,\ C)))\ ELSE\ Gf\ F\ C\ ex')$
$|\ TY\ ex\ ty \Rightarrow (let\ ex'{=}substE\ mp\ ex\ in\ (Ty\ ex\ ty,$
$IF\ ex' \doteq NewA\ 0\ THEN\ Cn\ (Bool\ ((Class\ Cl) = ty))\ ELSE\ Ty\ ex'\ ty)))\#mp)\ mp'\ es)\ ?\ x$
$= mp'\ ?\ x$

lemma *foldl-map-lookup''*:
$\bigwedge\ es.\ \forall\, a{\in}\ set\ es.\ x{\neq}f\ a \implies \forall\, mp'.\ (foldl\ (\lambda mp\ a.\ (f\ a,\ g\ mp\ a)\#mp)\ mp'\ es)\ ?\ x = mp'\ ?\ x$

lemma *getGfEx-size*:
$\bigwedge ex'\ .\ ex' \in set\ (getGfEx\ F\ C\ ex) \implies size\ ex' < size\ ex$

lemma *getGfEx-not-refl*:
$\bigwedge\ ex'.\ ex' \in set\ (getGfEx\ F\ C\ ex) \implies ex' \neq ex$
lemma *getHeapEx-GF-size*:
$\bigwedge F\ C\ ex\ .\ GF\ F\ C\ ex \in set\ (getHeapEx\ ex') \implies size\ ex < size\ ex'$

lemma *getHeapEx-TY-size*:
$\bigwedge ex\ ty\ .\ TY\ ex\ ty \in set\ (getHeapEx\ ex') \implies size\ ex < size\ ex'$

lemma *getHeapEx-GF-not-refl*:
$\bigwedge F\ C\ ex.\ GF\ F\ C\ ex \in set\ (getHeapEx\ ex') \Longrightarrow ex \neq ex'$

lemma *getHeapEx-TY-not-refl*:
$\bigwedge ex\ ty.\ TY\ ex\ ty \in set\ (getHeapEx\ ex') \Longrightarrow ex \neq ex'$

lemma *getGfEx-comp*:
$\bigwedge ex'.\ [\![\ ex' \in set\ (getGfEx\ F\ C\ ex)\ ]\!] \Longrightarrow \exists\, as\ bs\ cs.$
$((getGfEx\ F\ C\ ex) = as@(getGfEx\ F\ C\ ex')@bs@[ex']@cs \wedge$
$ex' \notin set\ (as@(getGfEx\ F\ C\ ex')@bs))$

lemma *getGFEx-mono*:
$\bigwedge ex''\ ex'.\ [\![\ ex'' \in set\ (getGfEx\ F\ C\ ex');\ ex' \in set\ (getGfEx\ F\ C\ ex)\ ]\!] \Longrightarrow \exists\, as\ bs\ cs.\ remdups'$ $(getGfEx\ F\ C\ ex) = as@[ex'']@bs@[ex']@cs$

lemma *getHeapEx-GF-comp*:
$GF\ F\ C\ ex' \in set\ (getHeapEx\ ex) \Longrightarrow \exists\, as\ bs\ cs.\ getHeapEx\ ex =$
$as\ @\ (getHeapEx\ ex')\ @\ bs\ @\ [GF\ F\ C\ ex']\ @\ cs \wedge$
$(GF\ F\ C\ ex') \notin set\ (as\ @\ (getHeapEx\ ex')\ @\ bs)$

lemma *getHeapEx-TY-comp*:
$TY\ ex'\ ty' \in set\ (getHeapEx\ ex) \Longrightarrow \exists\, as\ bs\ cs.\ getHeapEx\ ex =$
$as\ @\ (getHeapEx\ ex')\ @\ bs\ @\ [TY\ ex'\ ty']\ @\ cs \wedge$
$(TY\ ex'\ ty') \notin set\ (as\ @\ (getHeapEx\ ex')\ @\ bs)$

lemma *getHeapEx-mono-GF-GF*:
$[\![\ GF\ F''\ C''\ ex'' \in set\ (getHeapEx\ ex');\ GF\ F'\ C'\ ex' \in set\ (getHeapEx\ ex)\ ]\!]$
$\Longrightarrow \exists\, as\ bs\ cs.\ remdups'\ (getHeapEx\ ex) = as@[GF\ F''\ C''\ ex'']@bs@[GF\ F'\ C'\ ex']@cs$

lemma *getHeapEx-mono-GF-TY*:
$[\![\ GF\ F''\ C''\ ex'' \in set\ (getHeapEx\ ex');\ TY\ ex'\ ty \in set\ (getHeapEx\ ex)\ ]\!] \Longrightarrow \exists\, as\ bs\ cs.\ remdups'$ $(getHeapEx\ ex) = as@[GF\ F''\ C''\ ex'']@bs@[TY\ ex'\ ty]@cs$

lemma *getHeapEx-mono-TY-GF*:
$[\![\ TY\ ex''\ ty'' \in set\ (getHeapEx\ ex');\ GF\ F'\ C'\ ex' \in set\ (getHeapEx\ ex)\ ]\!]$
$\Longrightarrow \exists\, as\ bs\ cs.\ remdups'\ (getHeapEx\ ex) = as@[TY\ ex''\ ty'']@bs@[GF\ F'\ C'\ ex']@cs$

lemma *getHeapEx-mono-TY-TY*:
$[\![\ TY\ ex''\ ty'' \in set\ (getHeapEx\ ex');\ TY\ ex'\ ty' \in set\ (getHeapEx\ ex)\ ]\!]$
$\Longrightarrow \exists\, as\ bs\ cs.\ remdups'\ (getHeapEx\ ex) = as@[TY\ ex''\ ty'']@bs@[TY\ ex'\ ty']@cs$

1.3 Simulation between wpF and effS.

lemma *effS-wpF-Load*:
assumes *wf-Pi*: *wf* Π

assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-def*: $i = Load\ n$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *p'-domC*: $p' \in set\ (domC\ \Pi)$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(|lv{:=}I|))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(|lv{:=}I|))\ Q$

lemma *effS-wpF-Store*:
assumes *i-def*: $i = Store\ n$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(|lv{:=}I|))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(|lv{:=}I|))\ Q$

lemma *effS-wpF-Push*:
assumes *i-def*: $i = Push\ v$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$

assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: *check-instr' i P h stk loc C M pc frs*
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-New*:
assumes *i-def*: $i = New\ Cl$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h',fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: *check-instr' i P h stk loc C M pc frs*
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-Getfield*:
assumes *i-def*: $i = Getfield\ list1\ list2$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h',fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs$

e) else cs e|)
assumes *p′-def*: *p′* = *snd* (*snd fr′*)
assumes *check-i*: *check-instr′ i P h stk loc C M pc frs*
assumes *exec-i* : *exec-instr i P h stk loc C M pc frs* = σ′
assumes *Pi-def*: Π = (*P*,*An*)
shows ∀ *I*. *evalE* Π (*p*,σ,*e*(|*lv*:=*I*|)) (*wpF* Π *p p′ Q*) = *evalE* Π (*p′*,σ′,*e′*(|*lv*:=*I*|)) *Q*

lemma *effS-wpF-Putfield*:
assumes *i-def*: *i* = *Putfield list1 list2*
assumes *wf-Pi*: *wf* Π
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some i*
assumes *p-domC*: *p* ∈ *set* (*domC* Π)
assumes *i-instr*: *instrs-of P C M* ! *pc* = *i*
assumes *s-def*: *s* = (*p*,σ,*e*)
assumes *p-def*: *p* = (*C*,*M*,*pc*)
assumes *sigma-def*: σ = (*None*,*h*,(*stk*,*loc*,*p*)#*frs*)
assumes *s′-def*: *s′* = (*p′*,σ′,*e′*)
assumes *sigma′-def*: σ′ = (*None*,*h′*,*fr′*#*frs′*)
assumes *e′-def*: *e′* = *e*(|*cs* := *if* ∃ *M n*. *i* = *Invoke M n then h* # *cs e else if i* = *Return then tl* (*cs e) else cs e*|)
assumes *p′-def*: *p′* = *snd* (*snd fr′*)
assumes *check-i*: *check-instr′ i P h stk loc C M pc frs*
assumes *exec-i* : *exec-instr i P h stk loc C M pc frs* = σ′
assumes *Pi-def*: Π = (*P*,*An*)
shows ∀ *I*. *evalE* Π (*p*,σ,*e*(|*lv*:=*I*|)) (*wpF* Π *p p′ Q*) = *evalE* Π (*p′*,σ′,*e′*(|*lv*:=*I*|)) *Q*

lemma *effS-wpF-Checkcast*:
assumes *i-def*: *i* = *Checkcast Cl*
assumes *wf-Pi*: *wf* Π
assumes *handlesEx*: *handlesEx* (*fst* Π) *p′* = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some i*
assumes *p-domC*: *p* ∈ *set* (*domC* Π)
assumes *i-instr*: *instrs-of P C M* ! *pc* = *i*
assumes *s-def*: *s* = (*p*,σ,*e*)
assumes *p-def*: *p* = (*C*,*M*,*pc*)
assumes *sigma-def*: σ = (*None*,*h*,(*stk*,*loc*,*p*)#*frs*)
assumes *s′-def*: *s′* = (*p′*,σ′,*e′*)
assumes *sigma′-def*: σ′ = (*None*,*h*,*fr′*#*frs′*)
assumes *e′-def*: *e′* = *e*(|*cs* := *if* ∃ *M n*. *i* = *Invoke M n then h* # *cs e else if i* = *Return then tl* (*cs e) else cs e*|)
assumes *p′-def*: *p′* = *snd* (*snd fr′*)
assumes *check-i*: *check-instr′ i P h stk loc C M pc frs*
assumes *exec-i* : *exec-instr i P h stk loc C M pc frs* = σ′
assumes *Pi-def*: Π = (*P*,*An*)
shows ∀ *I*. *evalE* Π (*p*,σ,*e*(|*lv*:=*I*|)) (*wpF* Π *p p′ Q*) = *evalE* Π (*p′*,σ′,*e′*(|*lv*:=*I*|)) *Q*

lemma *effS-wpF-Invoke*:
assumes *i-def*: $i = Invoke\ Mn\ n$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s′-def*: $s' = (p',\sigma',e')$
assumes *sigma′-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e′-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p′-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$

assumes *p′-domC*: $p' \in set\ (domC\ \Pi)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-Return*:
assumes *i-def*: $i = Return$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s′-def*: $s' = (p',\sigma',e')$
assumes *sigma′-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e′-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p′-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
assumes *Pos-p*: $\Pi,(p,\sigma,e) \models Pos\ p$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-Pop*:
assumes *i-def*: $i = Pop$

assumes *wf-Pi*: *wf* Π
assumes *handlesEx*: *handlesEx* (*fst* Π) p' = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some i*
assumes *p-domC*: $p \in$ *set* (*domC* Π)
assumes *i-instr*: *instrs-of P C M* ! *pc* = *i*
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: σ = (*None*,*h*,(*stk*,*loc*,*p*)#*frs*)
assumes s'*-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: σ' = (*None*,*h*,*fr'*#*frs'*)
assumes e'*-def*: $e' = e$(|*cs* := *if* $\exists M\ n$. *i* = *Invoke M n then h* # *cs e else if i* = *Return then tl* (*cs e*) *else cs e*|)
assumes p'*-def*: p' = *snd* (*snd fr'*)
assumes *check-i*: *check-instr' i P h stk loc C M pc frs*
assumes *exec-i* : *exec-instr i P h stk loc C M pc frs* = σ'
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall$ *I*. *evalE* Π (p,σ,e(|*lv*:=*I*|)) (*wpF* Π *p* p' *Q*) = *evalE* Π (p',σ',e'(|*lv*:=*I*|)) *Q*

lemma *effS-wpF-IBin*:
assumes *i-def*: *i* = (*IBin no*)
assumes *wf-Pi*: *wf* Π
assumes *handlesEx*: *handlesEx* (*fst* Π) p' = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some i*
assumes *p-domC*: $p \in$ *set* (*domC* Π)
assumes *i-instr*: *instrs-of P C M* ! *pc* = *i*
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: σ = (*None*,*h*,(*stk*,*loc*,*p*)#*frs*)
assumes s'*-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: σ' = (*None*,*h*,*fr'*#*frs'*)
assumes e'*-def*: $e' = e$(|*cs* := *if* $\exists M\ n$. *i* = *Invoke M n then h* # *cs e else if i* = *Return then tl* (*cs e*) *else cs e*|)
assumes p'*-def*: p' = *snd* (*snd fr'*)
assumes *check-i*: *check-instr' i P h stk loc C M pc frs*
assumes *exec-i* : *exec-instr i P h stk loc C M pc frs* = σ'
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall$ *I*. *evalE* Π (p,σ,e(|*lv*:=*I*|)) (*wpF* Π *p* p' *Q*) = *evalE* Π (p',σ',e'(|*lv*:=*I*|)) *Q*

lemma *effS-wpF-Goto*:
assumes *i-def*: *i* = *Goto t*
assumes *wf-Pi*: *wf* Π
assumes *handlesEx*: *handlesEx* (*fst* Π) p' = *None*
assumes *cmd-p*: *cmd* Π *p* = *Some i*
assumes *p-domC*: $p \in$ *set* (*domC* Π)
assumes *i-instr*: *instrs-of P C M* ! *pc* = *i*
assumes *s-def*: $s = (p,\sigma,e)$

assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-CmpEq*:
assumes *i-def*: $i = CmpEq$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-IfIntCmp*:
assumes *i-def*: $i = IfIntCmp\ ro\ t$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$

assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF-IfFalse*:
assumes *i-def*: $i = IfFalse\ t$
assumes *wf-Pi*: $wf\ \Pi$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = None$
assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *sigma'-def*: $\sigma' = (None,h,fr'\#frs')$
assumes *e'-def*: $e' = e(\!|cs := if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\ \#\ cs\ e\ else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
assumes *p'-def*: $p' = snd\ (snd\ fr')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = \sigma'$
assumes *Pi-def*: $\Pi = (P,An)$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$
lemma *effS-wpF-Except*:
assumes *wf-Pi*: $wf\ \Pi$
assumes *sys-xptn-inv*: $\forall\ C \in sys\text{-}xcpts.\ (\exists ob.\ (h\ (addr\text{-}of\text{-}sys\text{-}xcpt\ C) = \lfloor ob \rfloor \wedge obj\text{-}ty\ ob = (Class\ C)))$
assumes *handlesEx*: $handlesEx\ (fst\ \Pi)\ p' = Some\ cn$
assumes *xa-sub-cn*: $P \vdash (cname\text{-}of\ h\ xa)\ \preceq^*\ cn$

assumes *cmd-p*: $cmd\ \Pi\ p = Some\ i$
assumes *p-domC*: $p \in set\ (domC\ \Pi)$
assumes *i-instr*: $instrs\text{-}of\ P\ C\ M\ !\ pc = i$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *p-def*: $p = (C,M,pc)$
assumes *sigma-def*: $\sigma = (None,h,(stk,loc,p)\#frs)$
assumes *s'-def*: $s' = (p',\sigma',e')$
assumes *check-i*: $check\text{-}instr'\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs$
assumes *exec-i* : $exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = (\lfloor xa \rfloor,\ h',\ frs'')$
assumes *findhandler-s*: $find\text{-}handler\ P\ xa\ h\ ((stk,\ loc,\ p)\ \#\ frs) = \sigma'$
assumes *sigma'-def*: $\sigma' = (None,\ h,\ ([Addr\ xa],\ loc',\ p')\ \#\ frs')$
assumes *e'-def*: $e' = e(\!|cs := drop\ (length\ frs - length\ frs')\ (cs\ e)|\!)$
assumes *Pi-def*: $\Pi = (P,An)$
assumes *Pos-p*: $\Pi,(p,\sigma,e) \models Pos\ p$

shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\sigma',e'(\!|lv{:=}I|\!))\ Q$

lemma *effS-wpF*:
assumes *wf-Pi*: $wf\ \Pi$
assumes *s-def*: $s = (p,\sigma,e)$
assumes *s′-def*: $s' = (p',\sigma',e')$

assumes *s-inv-Pos*: $\Pi,s \models inv\text{-}Pos\ \Pi\ (fst\ s)$
assumes *s-inv-Ty*:$\Pi,s \models inv\text{-}Ty\ \Pi\ (fst\ s)$
assumes *s-inv-ExTys*:$\Pi,s \models inv\text{-}ExTys\ \Pi\ (fst\ s)$
assumes *p′-B-succsF-p*: $(p',B) \in set\ (succsF\ \Pi\ p)$
assumes *s-B*: $\Pi,s \models B$
assumes *s-s′-effS*: $(s,s') \in effS\ \Pi$
shows $\forall\ I.\ evalE\ \Pi\ (p,\sigma,e(\!|lv{:=}I|\!))\ (wpF\ \Pi\ p\ p'\ Q) = evalE\ \Pi\ (p',\ \sigma',e'(\!|lv{:=}I|\!))\ Q$
end