1 Extended Jinja Virtual Machine

theory *JBC-Semantics* = *Form*:

setup ⟪ [*Simplifier.change-simpset-of op setmksimps* (*setmp proofs 2* (*mksimps mksimps-pairs*))] ⟫

1.1 States and Program Syntax

record *env* =
cs::*heap list*
lv::*var* ⇒ *val option*

types
jbc-state = *pos* × *jvm-state* × *env*

types
jbc-prog = *jvm-prog* × (*pos* ~~> *expr*)

1.2 Auxiliary functions

constdefs
incA::*pos* ⇒ *pos*
incA ≡ λ(*C*,*M*,*n*). (*C*,*M*,*n*+*1*)

constdefs
cmd::*jbc-prog* ⇒ *pos* ⇒ *instr option*
cmd ≡ λ(*P*,*A*) (*C*,*M*,*pc*). *case class P C of None* ⇒ *None*
| *Some c* ⇒ (*case* (*map-of* (*snd* (*snd c*)) *M*)
    of None ⇒ *None*
    | *Some m* ⇒ (*let is*=*fst* (*snd* (*snd* (*snd* (*snd m*))))
        in (*if pc* < *length is then Some* (*is*!*pc*) *else None*)))

consts
domMC::(*cname* × *mname* × *jvm-method*) ⇒ *pos list*

recdef *domMC measure* (λ*x*.*0*)
domMC (*C*,*M*,(*mxs*,*mxl*,*bd*,*et*)) = *map* (λ*n*. (*C*,*M*,*n*)) (*upt 0* (*length bd*))

consts
domCC::*jvm-method cdecl* ⇒ *pos list*

recdef *domCC measure* (λ(*C*,*cd*). *length* (*snd* (*snd cd*)))
domCC (*C*,(*C′*,*fs*,[])) = []
domCC (*C*,(*C′*,*fs*,(*M*,*Ts*,*T*,*m*)#*ms*)) = *domMC* (*C*,*M*,*m*) @ (*domCC* (*C*,(*C′*,*fs*,*ms*)))

constdefs
$domC$::jbc-$prog$ $\Rightarrow$ pos $list$
$domC$ $\Pi \equiv concat$ (map $domCC$ (fst Π))

constdefs
anF::jbc-$prog \Rightarrow pos \Rightarrow expr$ $option$
$anF \equiv \lambda(P,An)$ p. An ? p

consts
$hasAn$::jbc-$prog \Rightarrow pos \Rightarrow bool$
defs $hasAn$-def [$simp$]:
$hasAn$ Π $p \equiv$ ($case$ anF Π p of $None \Rightarrow False$ | $Some$ $a \Rightarrow True$)

constdefs aF::jbc-$prog \Rightarrow pos \Rightarrow expr$
aF Π $p \equiv$ ($case$ anF Π p of $None \Rightarrow TT$ | $Some$ $A \Rightarrow A$)

constdefs
$domA$ :: jbc-$prog \Rightarrow pos$ $list$
$domA \equiv$ (λ Π.[$p\in$ $domC$ Π. $hasAn$ Π p])

consts
$invokes$::$instr$ $option \times pos \Rightarrow bool$

recdef $invokes$ {}
— invokes (ins,(C,M,p)) checks whether ins calls method M.
$invokes$ ($Some$ ($Invoke$ Mn n),(C,M,pc)) = (Mn=M)
$invokes$ ($instr$,p) = $False$

constdefs
$callers$::jbc-$prog \Rightarrow pos \Rightarrow pos$ $list$
$callers$ Π $p \equiv$ [$cp \in$ ($domC$ Π). $invokes$ ((cmd Π cp),p)]

constdefs $classnames$ ::jvm-$prog \Rightarrow cname$ $list$
$classnames$ $P \equiv map$ fst P

constdefs has-$method$::jvm-$prog \Rightarrow cname \Rightarrow mname \Rightarrow bool$
has-$method$ P C $M \equiv$ (C,M) mem ($methodnames$ P)

1.3 Operational Semantics

constdefs
$Start$ :: $cname$
$Start \equiv$ ''$Start$''
$main$ ::$mname$
$main \equiv$ ''$main$''

constdefs $ipc::jbc\text{-}prog \Rightarrow pos$
$ipc\ P \equiv (Start,main,0)$

1.4 Transition Relation

consts
$effS::jbc\text{-}prog \Rightarrow (jbc\text{-}state \times jbc\text{-}state)\ set$

— Note: In case of an uncaught exception this semantics halts, whereas the JVM Semantics halts and empties the framestack.
inductive $effS\ \Pi$
intros

expt:
$[\![\ P = fst\ \Pi;\ p = (C,M,pc);\ i = instrs\text{-}of\ P\ C\ M\ !\ pc;$
$\quad \sigma = (None,h,(stk,loc,p)\#frs);$
$\quad has\text{-}method\ P\ C\ M;$
$\quad exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = (Some\ xa,h',bla);$
$\quad find\text{-}handler\ P\ xa\ h\ ((stk,loc,p)\#frs) = \sigma';$
$\quad \sigma' = (None,h,([Addr\ xa],loc',p')\#frs');$
$\quad e' = e(\!|cs:=drop\ (length\ frs - length\ frs')\ (cs\ e)|\!)$
$]\!] \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in effS\ \Pi$

nrml:
$[\![\ P = fst\ \Pi;\ p = (C,M,pc);\ i = instrs\text{-}of\ P\ C\ M\ !\ pc;$
$\quad \sigma = (None,h,(stk,loc,p)\#frs);$
$\quad has\text{-}method\ P\ C\ M;$
$\quad exec\text{-}instr\ i\ P\ h\ stk\ loc\ C\ M\ pc\ frs = (None,h',fr'\#frs');$
$\quad \sigma' = (None,h',fr'\ \#\ frs');$
$\quad p' = snd\ (snd\ fr');$
$\quad e' = e(\!|cs:=if\ \exists M\ n.\ i = Invoke\ M\ n\ then\ h\#(cs\ e)$
$\qquad else\ if\ i = Return\ then\ tl\ (cs\ e)\ else\ cs\ e|\!)$
$]\!] \Longrightarrow ((p,\sigma,e),(p',\sigma',e')) \in effS\ \Pi$

constdefs
$initS\ ::jbc\text{-}prog \Rightarrow jbc\text{-}state\ set$
$initS\ \Pi \equiv \{\ (p,\sigma,e)\ .\ p = ipc\ \Pi \wedge \sigma = start\text{-}state\ (fst\ \Pi)\ (fst\ (ipc\ \Pi))\ (fst\ (snd\ (ipc\ \Pi))) \wedge cs\ e =$
$[]\ \}$

consts $Reachables::jbc\text{-}prog \Rightarrow jbc\text{-}state\ set$

inductive *Reachables* Π
intros
init: $[\![\ s \in initS\ \Pi\ ]\!] \Longrightarrow s \in Reachables\ \Pi$
step: $[\![\ s \in Reachables\ \Pi;\ (s,s') \in effS\ \Pi\ ]\!]$
$\Longrightarrow s' \in Reachables\ \Pi$

1.5 Auxiliary Lemmas

lemma *sees-field-class*:
$P \vdash Cl\ sees\ F{:}T\ in\ Cl' \Longrightarrow is\text{-}class\ P\ Cl$

lemma *domCC-split*:
$domCC\ (C,(S,fs,ms@ms')) = domCC\ (C,(S,fs,ms))\ @\ domCC\ (C,(S,fs,ms'))$

lemma *cmd-domC*:
$\bigwedge p\ \Pi\ instr.\ \ cmd\ \Pi\ p = Some\ instr \Longrightarrow p \in set\ (domC\ \Pi)$

lemma *domC-cmd*:
$p \notin set\ (domC\ \Pi) \Longrightarrow\ cmd\ \Pi\ p = None$

lemma *domCC-split-parts*:
$(C,M,pc) \in set\ (domCC\ (C',S,Fs,Ms)) \Longrightarrow$
$\exists\ Ms'\ Ms''\ Ts\ T\ mxs\ mxl\ is\ et.\ Ms = Ms'\ @\ (M,Ts,T,(mxs,mxl,is,et))\ \#\ Ms''$
$\wedge\ (C,M,pc) \in set\ (domMC\ (C',M,mxs,mxl,is,et))$

lemma *domCC-map*:
$domCC\ (C,C',fs,ms) = concat\ (map\ (\lambda\ (M,Ts,T,m).\ domMC\ (C,M,m))\ ms)$

lemma *domMC-props*:
$\bigwedge\ m.\ (C,M,pc) \in set\ (domMC\ (C',M',m)) \Longrightarrow C = C' \wedge M = M' \wedge pc < length\ (fst\ (snd\ (snd\ m)))$

lemma *domA-simp*:
$domA = (\lambda\Pi.\ [pc{\in}domC\ \Pi.\ anF\ \Pi\ pc \neq None])$

lemma *effS-Reachables*:
$s \in Reachables\ \Pi = (\exists\ s0.\ s0 \in initS\ \Pi \wedge (s0,s) \in (effS\ \Pi)^*)$
lemma *check-inv*:
assumes *wt*:$wf\text{-}jvm\text{-}prog_k\ P$
assumes *initial*:$\sigma = start\text{-}state\ P\ C\ M \wedge (P \vdash C\ sees\ M{:}[]{\rightarrow}T = m\ in\ C)$
assumes *reachable*:$P \vdash \sigma\ {-jvm\rightarrow}\ \sigma'$
shows *check* $P\ \sigma'$
end