theory *SALSemantics* = *SALSyntax*:

1 SAL Semantics

datatype
  tval = *ILLEGAL* | *NAT nat* | *RA pos*

constdefs
  lift :: (*val* ⇒ *val* ⇒ *val*) ⇒ *tval* ⇒ *tval* ⇒ *tval*
 lift f a b ≡ (*case a of*
    ILLEGAL ⇒ *ILLEGAL*
  | *NAT m* ⇒ (*case b of*
      ILLEGAL ⇒ *ILLEGAL*
    | *NAT n* ⇒ *NAT* (*f m n*)
    | *RA ra* ⇒ *ILLEGAL*)
  | *RA ra* ⇒ *ILLEGAL*)

types
  tram = *loc* ⇒ *tval*

record *env* =
  cs :: (*nat* × *tram*) *list*
  h :: *pos list*

types
  SALstate = *pos* × *tram* × *env*
  SALform = *SALstate* ⇒ *bool*
  SALprocedure = *pname* × ((*instr* × (*SALform option*)) *list*)
  SALprogram = *SALprocedure list*

consts
  noProc :: *SALprogram* ⇒ *pname* ⇒ *bool*
primrec
 noProc [] *pn* = *True*
 noProc (*p* # *ps*) *pn* = (*let* (*pn*′, *bdy*) = *p in* (*if pn*′ = *pn then False else noProc ps pn*))

consts
  cmd :: *SALprogram* ⇒ (*pos* ⇒ *instr option*)
primrec
 cmd [] *pc* = *None*
 cmd (*p* # *ps*) *pc* = (*let* (*pn*, *bdy*) = *p*; (*pn*′, *i*) = *pc in*
    if pn = *pn*′ *then*
     (*if* (*i* < *length bdy* & (*noProc ps pn*))
      then Some (*fst* (*bdy*!*i*))
      else None)
    else (*cmd ps pc*))

consts
  $domC :: SALprogram \Rightarrow (pos\ list)$
primrec
 $domC\ [] = []$
 $domC\ (p\ \#\ ps) = (let\ (pn,\ bdy) = p\ in$
   $((if\ (noProc\ ps\ pn)\ then\ (map\ (\lambda i.\ (pn,\ i))\ (upt\ 0\ (length\ bdy)))\ else\ [])\ @$
$(delPos\ (domC\ ps)\ pn)))$

consts
  $anF :: SALprogram \Rightarrow (pos \Rightarrow SALform\ option)$
primrec
 $anF\ []\ pc = None$
 $anF\ (p\ \#\ ps)\ pc = (let\ (pn,\ bdy) = p;\ (pn',\ i) = pc\ in$
   $if\ pn = pn'\ then$
    $(if\ (i < length\ bdy\ \&\ (noProc\ ps\ pn))$
     $then\ snd\ (bdy!i)$
     $else\ None)$
   $else\ (anF\ ps\ pc))$

constdefs
  $step :: SALprogram \Rightarrow SALstate \Rightarrow SALstate\ option$
 $step\ p \equiv (\lambda((pn,i),\ m,\ e).\ (case\ (cmd\ p\ (pn,i))\ of$
   $None \Rightarrow None$
  $|\ Some\ ins \Rightarrow$
   $(case\ ins\ of$
    $SET\ x\ n \Rightarrow Some\ ((pn,\ i + 1),\ m[x \mapsto NAT\ n],\ e\ (|h := (h\ e)@[(pn,\ i)]|))$
   $|\ ADD\ x\ y \Rightarrow Some\ ((pn,\ i + 1),\ m[x \mapsto lift\ (op\ +)\ (m\ x)\ (m\ y)],\ e\ (|h :=$
$(h\ e)@[(pn,\ i)]|))$
   $|\ SUB\ x\ y \Rightarrow Some\ ((pn,\ i + 1),\ m[x \mapsto lift\ (op\ -)\ (m\ x)\ (m\ y)],\ e\ (|h :=$
$(h\ e)@[(pn,\ i)]|))$
   $|\ INC\ x \Rightarrow Some\ ((pn,\ i + 1),\ m[x \mapsto lift\ (op\ +)\ (m\ x)\ (NAT\ 1)],\ e\ (|h :=$
$(h\ e)@[(pn,\ i)]|))$
   $|\ JMPEQ\ x\ y\ t \Rightarrow (if\ (\exists\, n\ n'.\ m\ x = NAT\ n \wedge m\ y = NAT\ n')\ then\ (if\ (\exists\, n$
$n'.\ m\ x = NAT\ n \wedge m\ y = NAT\ n' \wedge n = n')$
     $then\ Some\ ((pn,\ i + t),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|))$
     $else\ Some\ ((pn,\ i + 1),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|)))$
    $else\ None)$
   $|\ JMPL\ x\ y\ t \Rightarrow (if\ (\exists\, n\ n'.\ m\ x = NAT\ n \wedge m\ y = NAT\ n')\ then\ (if\ (\exists\, n$
$n'.\ m\ x = NAT\ n \wedge m\ y = NAT\ n' \wedge n < n')$
     $then\ Some\ ((pn,\ i + t),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|))$
     $else\ Some\ ((pn,\ i + 1),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|)))$
    $else\ None)$
   $|\ JLE\ x\ y\ t \Rightarrow (if\ (\exists\, n\ n'.\ m\ x = NAT\ n \wedge m\ y = NAT\ n')\ then\ (if\ (\exists\, n\ n'.$
$m\ x = NAT\ n \wedge m\ y = NAT\ n' \wedge n \leq n')$
     $then\ Some\ ((pn,\ i + t),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|))$
     $else\ Some\ ((pn,\ i + 1),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|)))$
    $else\ None)$
   $|\ JMPB\ t \Rightarrow Some\ ((pn,\ i - t),\ m,\ e\ (|h := (h\ e)@[(pn,\ i)]|))$
    $|\ CALL\ x\ pn' \Rightarrow Some\ ((pn',\ 0),\ m[x \mapsto RA\ (pn,\ i + 1)],\ e\ (|h := (h$

e)@[(pn,i)], cs := (length (h e), m) # (cs e)⦈)
| RET x ⇒ (case (m x) of ILLEGAL ⇒ None | NAT n ⇒ None | RA ra ⇒ Some (ra, m, e ⦇h := (h e)@[(pn,i)], cs := tl (cs e)⦈))
| MOV s t ⇒ (case (m s)
of ILLEGAL ⇒ None
| NAT sa ⇒ (case (m t)
of ILLEGAL ⇒ None
| NAT ta ⇒ Some ((pn,i+1),m[ta ↦ (m sa)],e ⦇h := (h e)@[(pn, i)]⦈)
| RA ra ⇒ None
)
| RA ra ⇒ None
)
| HALT ⇒ None
)))

constdefs
effS :: *SALprogram* ⇒ ((*pos* × *tram* × *env*) × (*pos* × *tram* × *env*)) *set*
effS p ≡ {(*s*, *s*′). (*step p s* = *Some s*′)}

consts
initS :: *SALprogram* ⇒ *SALstate* ⇒ *bool*
safeS :: *SALprogram* ⇒ *SALstate* ⇒ *bool*

end