theory *SALSafetyLogic* = *TermCodegen* + *VerificationConditionGenerator* + *SALSemantics*:

1 SAL Safety Logic

constdefs
FalseF :: *SALform* (⌊*False*⌋)
FalseF ≡ λ*s*. *False*

constdefs
TrueF :: *SALform* (⌊*True*⌋)
TrueF ≡ λ *s*. *True*

constdefs
valid :: *SALprogram* ⇒ *SALstate* ⇒ *SALform* ⇒ *bool* ((-,- ⊨ -) [*61*,*61*,*60*] *60*)
valid prg s f ≡ *f*(*s*)

constdefs
Conj :: *SALform list* ⇒ *SALform* (⌊⋀⌋ - [*70*])
Conj fl ≡ λ *s*. ∀*f* ∈ *set fl*. *f*(*s*)

constdefs
Impl :: *SALform* ⇒ *SALform* ⇒ *SALform* (- ⌊⟹⌋ - [*61*,*60*] *60*)
Impl a b ≡ λ *s*. *a*(*s*) ⟶ *b*(*s*)

constdefs
isCall :: (*instr option*) ⇒ *pname* ⇒ *bool*
isCall instr pn' ≡ (*case instr of*
None ⇒ *False*
| *Some c* ⇒ (*case c of*
SET x n ⇒ *False*
| *ADD x y* ⇒ *False*
| *SUB x y* ⇒ *False*
| *INC x* ⇒ *False*
| *JMPEQ x y t* ⇒ *False*
| *JMPL x y t* ⇒ *False*
| *JLE x y t* ⇒ *False*
| *JMPB t* ⇒ *False*
| *CALL x pn* ⇒ (*pn* = *pn'*)
| *RET x* ⇒ *False*
| *MOV s t* ⇒ *False*
| *HALT* ⇒ *False*
))

constdefs
callpoints :: *SALprogram* ⇒ *pname* ⇒ *pos list*
callpoints prg pn ≡ [*cp* ∈ (*domC prg*). *isCall* (*cmd prg cp*) *pn*]

consts
ret-succs :: *SALprogram* ⇒ *pos* ⇒ *loc* ⇒ *pos list* ⇒ (*pos* × *SALform*) *list*
primrec
ret-succs prg pc x [] = []
ret-succs prg pc x (*cp* # *cps*) =
((*let*
(*pn*, *i*) = *pc*;
(*pn'*, *j*) = *cp*;
anF = (*case* (*anF prg cp*) *of*
None ⇒ *TrueF*
| *Some f* ⇒ λ (*pc*,*m*,*e*). *f* (*let*
(*k*,*m'*)=*hd* (*cs e*);
cs'=*tl* (*cs e*);
h' = *take k* (*h e*)
in ((*h e*)!*k*,*m'*,*e*(| *cs*:=*cs'*, *h*:=*h'*|))
)
)
in ((*pn'*, *Suc j*),
Conj [((λ (*pc*,*m*,*e*). (*m x*) = *RA* (*to-term pn'*, *Suc* (*to-term j*)) ∧ *pc*=(*pn*,*i*))::*SALform*),
anF])
)# (*ret-succs prg pc x cps*)
)

lemma *ret-succs-split*:
ret-succs prg pc x (*l1*@*l2*) = (*ret-succs prg pc x l1*) @ (*ret-succs prg pc x l2*)**done**

constdefs
succsF::*SALprogram* ⇒ *pos* ⇒ (*pos* × *SALform*) *list*
succsF prg pc ≡ (*let* (*pn*, *i*) = *pc in* (*case* (*cmd prg pc*) *of*
None ⇒ []
| *Some ins* ⇒ (*case ins of*
SET x n ⇒ [((*pn*, *i* + *1*), λ (*pc*,*m*,*e*). *pc*=(*pn*,*i*))]
| *ADD x y* ⇒ [((*pn*, *i* + *1*), λ (*pc*,*m*,*e*). *pc*=(*pn*,*i*))]
| *SUB x y* ⇒ [((*pn*, *i* + *1*), λ (*pc*,*m*,*e*). *pc*=(*pn*,*i*))]
| *INC x* ⇒ [((*pn*, *i* + *1*), λ (*pc*,*m*,*e*). *pc*=(*pn*,*i*))]
| *JMPEQ x y t* ⇒ [((*pn*, *i* + *t*),
λ(*pc*,*m*,*e*). (∃ *n n'*. *m x* = *NAT n* ∧ *m y* = *NAT n'* ∧ *n* = *n'*)
∧ *pc*=(*pn*,*i*)
),
((*pn*, *i* + *1*),
λ(*pc*,*m*,*e*). (∃ *n n'*. *m x* = *NAT n* ∧ *m y* = *NAT n'* ∧ *n* ≠ *n'*)
∧ *pc*=(*pn*,*i*)
)
]
| *JMPL x y t* ⇒ [((*pn*, *i* + *t*),
λ(*pc*,*m*,*e*). (∃ *n n'*. *m x* = *NAT n* ∧ *m y* = *NAT n'* ∧ *n* < *n'*)
∧ *pc*=(*pn*,*i*)
),

```
                  ((pn, i + 1),
                    λ(pc,m,e). (∃ n n'. m x = NAT n ∧ m y = NAT n' ∧ ¬ (n <
n')) ∧ pc=(pn,i)
                  )
                 ]
    | JLE x y t ⇒ [((pn, i + t),
                    λ(pc,m,e). (∃ n n'. m x = NAT n ∧ m y = NAT n' ∧ n ≤ n')
∧ pc=(pn,i)
                  ),
                  ((pn, i + 1),
                    λ(pc,m,e). (∃ n n'. m x = NAT n ∧ m y = NAT n' ∧ ¬ (n ≤
n')) ∧ pc=(pn,i)
                  )
                 ]
   | JMPB t ⇒ [((pn, i − t),  λ (pc,m,e). pc=(pn,i))]
   | CALL x pn' ⇒ [((pn', 0), λ (pc,m,e). pc=(pn,i))]
   | RET x ⇒ (ret-succs prg pc x (callpoints prg pn))
   | MOV s t ⇒ [((pn,i+1),λ (pc,m,e). pc=(pn,i))]
   | HALT ⇒ []

)))

constdefs
  wpF :: SALprogram ⇒ pos ⇒ pos ⇒ SALform ⇒ SALform
 wpF prg pc1 pc2 Q ≡ let (pn, i) = pc1; (pn',j) = pc2 in
    (case (cmd prg (pn,i)) of
      None ⇒ FalseF
    | Some ins ⇒ (case ins of
       SET x n ⇒ (λ(pc,m,e). Q ((pn', j), m[x ↦ NAT n], e(|h:=(h e)@[(pn,i)]|)))
      | ADD x y ⇒ (λ(pc,m,e). Q ((pn', j), m[x ↦ lift (op +) (m x) (m y)],
e(|h:=(h e)@[(pn,i)]|)))
     | SUB x y ⇒ (λ(pc,m,e). Q ((pn', j), m[x ↦ lift (op −) (m x) (m y)], e(|h:=(h
e)@[(pn,i)]|)))
      | INC x ⇒ (λ(pc,m,e). Q ((pn', j), m[x ↦ lift (op +) (m x) (NAT 1)],
e(|h:=(h e)@[(pn,i)]|)))
      | JMPEQ x y t ⇒ (λ(pc,m,e). Q ((pn', j),m,e(|h:=(h e)@[(pn,i)]|)))
      | JMPL x y t ⇒ (λ(pc,m,e). Q ((pn', j),m,e(|h:=(h e)@[(pn,i)]|)))
      | JLE x y t ⇒ (λ(pc,m,e). Q ((pn', j),m,e(|h:=(h e)@[(pn,i)]|)))
      | JMPB t ⇒ (λ(pc,m,e). Q ((pn', j),m,e(|h:=(h e)@[(pn,i)]|)))
       | CALL x pn'' ⇒ (λ (pc,m,e). Q ((pn', j), m[x↦RA (pn,Suc i)], e(|h:=(h
e)@[(pn,i)],cs:=(length (h e),m)#(cs e)|)))
      | RET x ⇒ (λ (pc,m,e). Q ((pn', j), m, e(|h:=(h e)@[(pn,i)],cs:=tl (cs e)|)))
      | MOV s t ⇒ (λ (pc,m,e). (case (m s)
                                of ILLEGAL ⇒ False
                                | NAT sa ⇒ (case (m t)
                                             of ILLEGAL ⇒ False
                                           | NAT ta ⇒ Q ((pn',j),m[ta ↦ (m sa)],e(|h:=(h
e)@[(pn,i)]|))
                                             | RA ra ⇒ False
```

```
                                    )
                                | RA ra ⇒ False
                               )
                  )
      | HALT ⇒ TrueF
))

constdefs
  MAX :: nat
 MAX ≡ 15

constdefs
  safeF :: SALprogram ⇒ pos ⇒ SALform
 safeF prg pc ≡ (let (pn,i) = pc in (case (cmd prg pc) of
    None ⇒ FalseF
  | Some ins ⇒ (case ins of
      SET x n ⇒ (λ (pc, m, e). n ≤ MAX)
    | ADD x y ⇒ (λ (pc, m, e). (∃ n. m x = NAT n) ∧ (∃ n. m y = NAT n) ∧
(∃ n. (lift (op +) (m x) (m y)) = NAT n ∧ n ≤ MAX))
    | SUB x y ⇒ (λ (pc, m, e). (∃ n. m x = NAT n) ∧ (∃ n. m y = NAT n))
    | INC x  ⇒ (λ (pc, m, e). ∃ n. m x = NAT n ∧ n < MAX)
    | JMPEQ x y t ⇒ (λ (pc, m, e). (∃ n. m x = NAT n) ∧ (∃ n. m y = NAT n))
    | JMPL x y t ⇒ (λ (pc, m, e). (∃ n. m x = NAT n) ∧ (∃ n. m y = NAT n))
    | JLE x y t ⇒ (λ (pc, m, e). (∃ n. m x = NAT n) ∧ (∃ n. m y = NAT n))
    | JMPB t ⇒ TrueF
    | CALL x pn' ⇒ TrueF
    | RET x ⇒ (λ (pc,m,e). (∃ pn' i'. (m x) = RA (pn', Suc i') ∧ (∃ k m' cl css.
(cs e) = (k, m') # (cl # css) ∧ (pn', i') = (h e)!k)))
    | MOV s t ⇒ (λ (pc,m,e). (∃ sa. (m s) = NAT sa) ∧ (∃ ta. (m t) = NAT ta))
    | HALT ⇒ TrueF
)))

constdefs
  ipc :: SALprogram ⇒ pos
  ipc prg ≡ (0,0)

constdefs
initF :: SALprogram ⇒ SALform
initF prg ≡ λ (pc,m,e). pc = (0, 0) ∧
                        cs e = [(0,m)] ∧
                        h e = [] ∧
                        (∀ X. (m X) = ILLEGAL)

lemma [code]:
initF prg = term (λ (pc,m,e).  pc = (0, 0) ∧
                               cs e = [(0,m)] ∧
                               h e = [] ∧
                              (∀ X. (m X) = ILLEGAL))
by (simp add: term-def initF-def)
```

constdefs
$isafeF$::$SALprogram \Rightarrow pos \Rightarrow SALform$
$isafeF\ prg\ pc \equiv isafe(domC\ prg,prg,anF\ prg,pc,FalseF,Conj,Impl,safeF,succsF,wpF)$

constdefs
$isafeP$::$SALprogram \Rightarrow SALstate\ set\ (isafe_{\square}\text{-}\ [70])$
$isafeP\ prg \equiv (isafeP'\ effS\ valid\ initF\ isafeF\ prg)$

lemma *isafeP-induct*:
$[\![\ s \in (isafeP\ prg);$
$\quad \bigwedge s.\ [\![\ valid\ prg\ s\ (initF\ prg)\ ]\!] \Longrightarrow P\ s;$
$\quad \bigwedge s\ s'.\ [\![\ s \in (isafeP\ prg);\ valid\ prg\ s\ (isafeF\ prg\ (fst\ s));\ valid\ prg\ s'\ (isafeF\ prg\ (fst\ s'));\ (s,s') \in (effS\ prg);\ P\ s\ ]\!] \Longrightarrow P\ s'\ ]\!] \Longrightarrow P\ s$**done**

constdefs
$provable :: SALprogram \Rightarrow SALform \Rightarrow bool\ ((\text{-} \vdash \text{-})\ [61,60]\ 60)$
$provable\ prg\ f \equiv \forall\ s.\ \ s \in (isafeP\ prg) \longrightarrow valid\ prg\ s\ f$

end