theory *SALTimePlatform* = *SALOverflowPlatform*:

1 SAL Overflow Platform

— The consumer platform regards programs as safe iff no overflow occurs.

1.1 Platform Definition

constdefs
MAXTIME :: *nat*
MAXTIME ≡ *20*

constdefs
tim::*env* ⇒ *nat*
tim e ≡ *length* (*h e*)

constdefs
safeFT :: *SALprogram* ⇒ *pos* ⇒ *SALform*
safeFT 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*). (*tim e*) < *MAXTIME*)
| *ADD x y* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *SUB x y* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *INC x* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *JMPEQ x y t* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *JMPL x y t* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *JLE x y t* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *JMPB t* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME* ∨ *t* = *0*)
| *CALL x pn′* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *RET x* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *MOV s t* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
| *HALT* ⇒ (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)
)
)
)

lemma [*code*]:
safeFT prg pc =
(*let* (*pn*,*i*) = *pc in*
(*case* (*cmd prg pc*)
of None ⇒ *FalseF*
| *Some ins* ⇒
(*case ins*
of SET x n ⇒ *term* (λ (*pc*, *m*, *e*). (*tim e*) < *MAXTIME*)

```
    | ADD x y ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | SUB x y ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | INC x  ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | JMPEQ x y t ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | JMPL x y t ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | JLE x y t ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | JMPB t ⇒ term (λ (pc, m, e). (tim e) < MAXTIME ∨ (to-term t)=0)
    | CALL x pn' ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | RET x ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | MOV s t ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
    | HALT ⇒ term (λ (pc, m, e). (tim e) < MAXTIME)
   )
  )
 )
apply (simp only: safeFT-def term-def Let-def split-def to-term-def)
done


constdefs
safeF:: SALprogram ⇒ pos ⇒ SALform
safeF prg pc ≡ Conj [(SALSafetyLogic.safeF prg pc),safeFT prg pc]


lemma[code]:
safeF prg pc ≡ let a=(SALSafetyLogic.safeF prg pc); b=(safeFT prg pc)
              in (term (Conj [(to-term a),(to-term b)]))
apply (simp only: safeF-def to-term-def term-def Let-def)
done

constdefs
  vcgSALT :: SALprogram ⇒ SALform
 vcgSALT prg ≡ vcG Conj Impl FalseF ipc initF safeF succsF wpF domC domA
anF prg


generate-code (vcgSALT.ML) [term-of]
  vcg = vcgSALT

constdefs
isafeF::SALprogram ⇒ pos ⇒ SALform
isafeF prg pc ≡ isafe(domC prg,prg,anF prg,pc,FalseF,Conj,Impl,safeF,succsF,wpF)


constdefs
isafeP::SALprogram ⇒ SALstate set (isafe□- [70])
isafeP prg ≡ (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
$\quad provable :: SALprogram \Rightarrow SALform \Rightarrow bool\ ((\text{-} \vdash \text{-})\ [61,60]\ 60)$
$\quad provable\ prg\ f \equiv \forall\ s.\ \ s \in (isafeP\ prg) \longrightarrow valid\ prg\ s\ f$
end