theory *EX-Sum = SALTypeSafetyFWInst*:

1 Example: Type Safety

We analyse a program that adds all numbers from 0 to N.

1.1 Variables

constdefs
N::*nat*
N≡*0*
I::*nat*
I≡*1*
S::*nat*
S≡*2*

— initial values
constdefs
N0::*nat*
N0≡*4*
I0::*nat*
I0≡*0*
S0::*nat*
S0≡*0*

1.2 Annotated program

constdefs
prog::*SALprogram*
prog ≡ [
(*SET N N0*,*None*),
(*SET I I0*,*None*),
(*SET S S0*,*None*),
(*JMPEQ I N 4*, *Some* (λ(*pc*, *m*). *m I* ≠ *ILLEGAL* ∧ *m N* ≠ *ILLEGAL* ∧ *m S* ≠ *ILLEGAL*)),
(*ADD S I*, *None*),
(*INC I*, *None*),
(*JMPB 3*, *None*),
(*JMPB 0*, *Some TrueF*)
]

lemma [*code*]:
prog = [
(*SET N N0*,*None*),
(*SET I I0*,*None*),
(*SET S S0*,*None*),

(*JMPEQ I N 4*, *Some* (*term* (λ(*pc*, *m*). *m* (*to-term I*) ≠ *ILLEGAL* ∧ *m* (*to-term N*) ≠ *ILLEGAL* ∧ *m* (*to-term S*) ≠ *ILLEGAL*))),
(*ADD S I*, *None*),
(*INC I*, *None*),
(*JMPB 3*, *None*),
(*JMPB 0*, *Some TrueF*)
]
apply (*simp add*: *term-def to-term-def prog-def*)
done

1.3 The verification condition

generate-code (*vcgTSAL.ML*) [*term-of*]
prg = *prog*
vcg = *vcgTSAL*

constdefs *vc*::*SALform*
vc ≡ %*s*. (%*s*. (%(*p*, *m*). *p* = *0* & (*ALL X*. *m X* = *SALSemantics.tval.ILLEGAL*)) *s* −−> (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL* & *m 0* ~= *SALSemantics.tval.ILLEGAL* & *m* (*Suc* (*Suc 0*)) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc 0*)), *SALSemantics.update m* (*Suc* (*Suc 0*)) (*SALSemantics.tval.NAT 0*))) *s*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc 0*), *SALSemantics.update m* (*Suc 0*) (*SALSemantics.tval.NAT 0*))) *s*) *s* & (%*s*. *True*) *s*) *s*) (*Suc 0*, *SALSemantics.update m 0* (*SALSemantics.tval.NAT* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))) *s*) *s* & (%*s*. *True*) *s*) *s*) *s* & (%*s*. (%*s*. (%*s*. (%*s*. (%*s*. (%*s*. *True*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL* & *m 0* ~= *SALSemantics.tval.ILLEGAL* & *m* (*Suc* (*Suc 0*)) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. *True*) *s*) *s*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) = *m 0*) *s* & (%*s*. *True*) *s*) *s*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. *True*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))), *m*)) *s*) *s* & (%*s*. (%*s*. (%*s*. (%*s*. (%*s*. *True*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL* & *m 0* ~= *SALSemantics.tval.ILLEGAL* & *m* (*Suc* (*Suc 0*)) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. *True*) *s*) *s*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) ~= *m 0*) *s* & (%*s*. *True*) *s*) *s*) *s* −−> (%(*pc*, *m*). (%*s*. (%(*p*, *m*). *m* (*Suc* (*Suc 0*)) ~= *SALSemantics.tval.ILLEGAL* & *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%(*p*, *m*). *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. (%*s*. *True*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%(*pc*, *m*). *m* (*Suc 0*) ~= *SALSemantics.tval.ILLEGAL* & *m 0* ~= *SALSemantics.tval.ILLEGAL* & *m* (*Suc* (*Suc 0*)) ~= *SALSemantics.tval.ILLEGAL*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc 0*)), *m*)) *s*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))), *SALSemantics.update m* (*Suc 0*) (*SALSemantics.lift op* + (*m* (*Suc 0*)) (*SALSemantics.tval.NAT 1*)))) *s*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))), *SALSemantics.update m* (*Suc* (*Suc 0*)) (*SALSemantics.lift op* + (*m* (*Suc* (*Suc 0*))) (*m* (*Suc 0*))))) *s*) *s* & (%*s*.

True) *s*) *s*) (*Suc* (*Suc* (*Suc* (*Suc 0*))), *m*)) *s*) *s* & (%*s*. *True*) *s*) *s*) *s* & (%*s*. (%*s*. (%*s*. (%*s*. (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. *True*) *s* & (%*s*. *True*) *s*) *s*) *s* & (%*s*. (%*s*. *True*) *s* & (%*s*. *True*) *s*) *s*) *s* −−> (%(*pc*, *m*). (%*s*. (%*s*. *True*) *s* & (%*s*. (%*s*. *True*) *s* & (%*s*. *True*) *s*) *s*) (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))), *m*)) *s*) *s* & (%*s*. *True*) *s*) *s* & (%*s*. *True*) *s*) *s*) *s*

lemma *all-ex-tval*: (∀ *x*. *m x* ~= *ILLEGAL*) ⟹ ∀ *x*. ∃ *k*. *m x* = *NAT k***done**

lemma *ex-tval*: (*m x* ~= *ILLEGAL*) ⟹ ∃ *k*. *m x* = *NAT k***done**

1.4 Program Verification

First, we check the program's wellformedness.

lemma *wf-prog*:
wf prog
apply (*simp add*: *wf-def checkPos.simps prog-def Let-def split-def fst-conv snd-conv cmd-def domC-def anF-def*)
done

Then we prove the verification condition. One simplifier call suffices.

lemmas *vc-simps* [*simp*] = *vc-def provable-def valid-def initF-def valid-def Let-def split-def fst-conv snd-conv update-def fun-upd-apply S-def I-def N-def S0-def I0-def N0-def lift-def all-ex-tval ex-tval*

lemma *vc-prog-holds*: *prog* ⊢ *vc*
apply (*auto simp add*: *vc-def tval.cases split add*: *tval.split tval.split-asm*)
done

Next, we check that the generated vc is correct

lemma *vc-correct*: *vcgTSAL prog* = *vc*
apply (*simp add*: *nat-number vcgTSAL-def prog-def vcG-def ipc-def wpF-def succsF-def safeF-def anF-def domC-def domA-def isafe.simps cmd-def Conj-def Impl-def TrueF-def FalseF-def*)
done

theorem *prog-isSafe*:
isSafe prog
apply (*rule platform-soundness*)
apply (*rule wf-prog*)
apply (*simp only*: *vc-correct*)
apply (*rule vc-prog-holds*)
done

end