theory *SALOverflowFWInst-deep* = *SALOverflowPlatform-deep* + *AuxBox2*:

In this theory we prove that our instantiated functions satisfy the PCC Framework's requirements. In the very end this leads to a reuse of the VCG Soundness theorem and thus guarantees the soundness of our platform

1 Requirements for the Safety Logic hold!

theorem *SALSafetyLogicIns*:
*SafetyLogic Conj Impl TrueF FalseF valid***done**

2 Auxiliary Lemmas

lemma *checkPos-split*:
checkPos prg (*l1*@*l2*) = ((*checkPos prg l1*) ∧ (*checkPos prg l2*))**done**

lemma *isafe-imp-safeF*:
∀ *prg s*. *valid prg s* (*isafe* (*domC prg*, *prg*, *anF prg*, *fst s*, *FalseF*, *Conj*, *Impl*, *safeF*, *succsF*, *wpF*)) ⟶ *valid prg s* (*safeF prg* (*fst s*))**done**

lemma *isafeF-imp-safeF*:
valid prg (*p*,*m*,*e*) (*isafeF prg p*) ⟹ *valid prg* (*p*,*m*,*e*) (*safeF prg p*)**done**

2.1 Pre Safe States

We define the set of so called pre safe states. These are states that originate from a (previously) safe execution.

consts
safeP::*SALprogram* ⇒ *SALstate set*

inductive *safeP prg*
intros
init: ⟦ *valid prg* (*p*,*m*,*e*) (*initF prg*) ⟧ ⟹ (*p*,*m*,*e*) ∈ (*safeP prg*)
step: ⟦ (*p*,*m*,*e*) ∈ (*safeP prg*);
valid prg (*p*,*m*,*e*) (*safeF prg p*);
valid prg (*p'*,*m'*,*e'*) (*safeF prg p'*);
((*p*,*m*,*e*),(*p'*,*m'*,*e'*)) ∈ (*effS prg*) ⟧
⟹ (*p'*,*m'*,*e'*) ∈ (*safeP prg*)

lemma *isafeP-imp-safeP*:
(*p*,*m*,*e*) ∈ *isafeP prg* ⟹ (*p*,*m*,*e*) ∈ *safeP prg***done**

3 System Invariants

Useful properties about states that originate from a safe execution.

3.1 System Invariant 1

consts *sysinv*::*SALstate* × *SALprogram* ⇒ *bool*

recdef
sysinv *measure* (λ ((*pc*,*m*,*e*),*prg*). *length* (*cs e*))
sysinv ((*pc*,*m*,*e*),*prg*) = (*case* (*cs e*)
of [] ⇒ *False*
| *c*#*css* ⇒ (*let* (*k*,*m*)=*c*; (*pn*′,*i*′)=(*h e*)!*k*; (*pn*,*i*)=*pc*
in (*case css*
of [] ⇒ *pn*=*0* ∧ (∀ *i0 x*. *cmd prg* (*0*,*i0*) ≠ *Some* (*RET x*))
| *c*′#*css*′ ⇒ (∃ *x*. *cmd prg* (*pn*′,*i*′) = *Some* (*CALL x pn*))
∧ *k* < *length* (*h e*)
∧ *sysinv* (((*pn*′,*i*′),*m*,*e*(|*cs*:=*css*,*h*:=*take* *k* (*h e*)|)),*prg*)
)
)
)
(**hints** *recdef-simp*: *filterreduction*)

lemma *sysinv-calltime*:
⟦ *sysinv* ((*p*,*m*,*e*),*prg*) ⟧ ⟹ *1* < *length* (*cs e*) ⟶ *fst* (*hd* (*cs e*)) < (*length* (*h e*))**done**

theorem *sysinv-pres*:
⋀ *prg p m e*. ⟦ *wf prg* ; (*p*,*m*,*e*) ∈ (*safeP prg*) ⟧ ⟹ *sysinv*((*p*,*m*,*e*),*prg*)**done**

lemma *sysinv-pres*′:
⋀ *prg s*. ⟦ *wf prg*; *s* ∈ *isafeP prg* ⟧ ⟹ *sysinv* (*s*,*prg*)**done**

3.2 System Invariant 2

consts *sysinv2*::*SALstate* × *SALprogram* ⇒ *bool*

recdef
sysinv2 *measure* (λ ((*pc*,*m*,*e*),*prg*). *length* (*cs e*))
sysinv2 ((*pc*,*m*,*e*),*prg*) = (*case* (*cs e*)
of [] ⇒ *False*
| *c*#*css* ⇒ (*let* (*k*,*m*′′)=*c*; (*pn*′,*i*′)=(*h e*)!*k*
in (*case css*
of [] ⇒ *True*
| *c*′#*css*′ ⇒ (*k* < *length* (*h e*) ∧ *valid prg* ((*pn*′,*i*′),*m*′′,*e*(| *cs*:=*css*, *h*:=*take k* (*h e*) |)) (*isafeF prg* (*pn*′,*i*′)) ∧
sysinv2 (((*pn*′,*i*′),*m*′′,*e*(| *cs*:=*css*, *h*:=*take k* (*h e*) |)),*prg*)
)
)
))

(**hints** *recdef-simp*: *filterreduction*)

lemma *sysinv2-pres*:
$\forall$ *prg s*. *wf prg* $\longrightarrow$ *s* $\in$ (*isafeP prg*) $\longrightarrow$ *sysinv2* (*s*,*prg*)**done**

lemma *ex-weak*: $\exists\ n\ n'.\ a = NAT\ n \wedge b = NAT\ n' \wedge n = n' \Longrightarrow \exists\ n\ n'.\ a = NAT\ n \wedge b = NAT\ n'$
apply *auto*
done

lemma *ex-eq*: $(\exists n.\ m\ nat1 = NAT\ n) \wedge (\exists n'.\ m\ nat2 = NAT\ n') \Longrightarrow \neg (\exists n.\ m\ nat1 = NAT\ n \wedge m\ nat2 = NAT\ n) \Longrightarrow (\exists n\ n'.\ m\ nat1 = NAT\ n \wedge m\ nat2 = NAT\ n' \wedge n \neq n')$
apply *auto*
done

lemma *ex-less*: $(\exists n.\ m\ nat1 = NAT\ n) \wedge (\exists n'.\ m\ nat2 = NAT\ n') \Longrightarrow \neg (\exists n.\ m\ nat1 = NAT\ n \wedge (\exists n'.\ m\ nat2 = NAT\ n' \wedge n < n')) \Longrightarrow \exists n.\ m\ nat1 = NAT\ n \wedge (\exists n'.\ m\ nat2 = NAT\ n' \wedge \neg\ n < n')$
apply *auto*
done

lemma *list-length-nth*:
(*la*@[*i*,*j*])!(*length la*)=*i***done**

lemma *list-length-nth2*:
(*la*@[*i*])!(*length la*)=*i***done**

lemma *list-length-suc-nth*:
(*la*@[*i*,*j*])!(*Suc* (*length la*))=*j***done**

lemma *path-succsF*:
⟦ *l* $\in$ *paths prg succsF*; *k*+*1*<*length l* ⟧ $\Longrightarrow$ $\exists$ *B*. (*l*!(*k*+*1*),*B*) $\in$ *set* (*succsF prg* (*l*!*k*))**done**

lemma *less-chain-simp*:
⟦ $\forall$ *k*. *length l* <= *k*+*1* $\vee$ *l*!*k* < *l*!(*k*+*1*); *1* < *length* (*l*::*pos list*) ⟧ $\Longrightarrow$ *hd l* < *last l***done**

lemma *path-length*:
⟦ *l* $\in$ *paths prg succsF* ⟧ $\Longrightarrow$ *1* < *length l***done**

lemma *loop-has-back-jump*:
⟦ *l* $\in$ *paths prg succsF*; *hd l* = *last l* ⟧ $\Longrightarrow$ $\exists$ *k*. (*k*+*1*)<*length l* $\wedge$ *l*!(*k*+*1*) <= *l*!*k***done**

lemma *back-jumps-annotated*:
wf prg $\Longrightarrow$ $\forall$ *pc''* *pc* *B*. (*pc''*,*B*) $\in$ *set* (*succsF prg pc*) $\longrightarrow$ *pc''*<=*pc* $\longrightarrow$ (*anF prg pc''*) $\neq$ *None***done**

lemma *isafeP-isafeF-initF*:
s $\in$ (*isafeP prg*) $\Longrightarrow$ *valid prg s* (*initF prg*) $\vee$ *valid prg s* (*isafeF prg* (*fst s*))
apply (*erule isafeP-elims*)
apply *simp+*
done

lemma *lookup-changedvars*:
mp ? (*V v*) = *Some e* $\Longrightarrow$ (*changedvars mp*) ? *v* = *Some e***done**

lemma *lookup-changedvars-neg*:
mp ? (*V v*) = *None* $\Longrightarrow$ (*changedvars mp*) ? *v* = *None*