1 Annotated Control Flow Paths

theory *CFG = SafetyPolicy + AuxBox*:

1.1 Annotated Control Flow Graphs

consts *enoughAn*::(*'prog* ⇒ *'pos* ⇒ (*'pos*×*'form*) *list*) ⇒ *'prog* ⇒ (*'prog* ⇒ *'pos* ⇒ *'form option*) ⇒ *bool*

locale *CFG = SafetyPolicy +*

fixes *domC*:: *'prog* ⇒ *'pos list*
fixes *ipc*:: *'prog* ⇒ *'pos*

fixes *anF*:: *'prog* ⇒ (*'pos* ⇒ *'form option*)
fixes *aF*::*'prog* ⇒ *'pos* ⇒ *'form*
defines *aF* ≡ λ Π *p*. (*case anF* Π *p of None* ⇒ ⌊*True*⌋ | *Some An* ⇒ *An*)
fixes *domA*:: *'prog* ⇒ *'pos list*
defines *domA* ≡ λ Π. [*pc*∈ *domC* Π. (*anF* Π *pc*) ≠ *None*]

fixes *succsF*:: *'prog* ⇒ *'pos* ⇒ (*'pos* × *'form*) *list*

fixes *wf*:: *'prog* ⇒ *bool*

assumes *wf-anF*:
wf Π ⟹ *enoughAn succsF* Π *anF*

fixes *correctAn*::*'prog* ⇒ *bool*
defines *correctAn* ≡ λ Π. ∀ *s* ∈ *Reachables* Π. Π,*s* ⊨ *aF* Π (*fst s*)

fixes *saF*::*'prog* ⇒ *'pos* ⇒ *'form*
defines
saF ≡ λ Π *p*. ⌊⋀⌋ [*safeF* Π *p*, *aF* Π *p*]

1.2 Auxiliary definitions

This theory defines notions for paths and loops in a program. These are required to state the wellformedness condition wf Pi, which checks wether a program has at least one annotation per loop.

1.3 Index of list elements

consts
idx::*'a list* ⇒ *'a* ⇒ *nat*

primrec
idx [] *a* = *0*

idx (*l*#*ls*) *a* = (*if l*=*a*
then 0
else Suc (*idx ls a*)
)

1.4 Paths

The set of all control flow paths of a given program**consts**
paths::('*a* ⇒ ('*a* × '*b*) *list*) ⇒ ('*a list*) *set*
inductive (*paths sc*)
intros
start: ⟦ (*p*',*B*) ∈ *set* (*sc p*) ⟧ ⟹ [*p*,*p*'] ∈ (*paths sc*)
step: ⟦ *l*@[*p*] ∈ (*paths sc*); (*p*',*B*) ∈ *set* (*sc p*) ⟧ ⟹ *l*@[*p*]@[*p*'] ∈ (*paths sc*)

1.5 Cycles

A path is a cycle if the first position equals the last position.**constdefs**
isCycle::('*prog* ⇒ '*pos* ⇒ ('*pos*×'*form*) *list*) ⇒ '*prog* ⇒ ('*pos list*) ⇒ *bool*
isCycle succsF Π *l* ≡ (*l*∈ *paths* (*succsF* Π) ∧ (*hd l* = *last l*))

constdefs *noAnn*::'*pos list* ⇒ ('*pos* ⇒ '*form option*) ⇒ *bool*
noAnn l pan ≡ (∀ *p*. *p*∈ *set l* ⟶ *pan p* = *None*)

defs *enoughAn-def*:
enoughAn succsF Π *an* ≡ (∀ *l*. *isCycle succsF* Π *l* ⟶ ¬ (*noAnn l* (*an* Π)))

constdefs *noAnnOnWay*::('*prog* ⇒ '*pos* ⇒ ('*pos*×'*form*) *list*) ⇒
('*prog* ⇒ '*pos* ⇒ '*form option*) ⇒ '*prog* ⇒ '*pos* ⇒ '*pos* ⇒ *bool*
noAnnOnWay succsF an Π *pc pc*' ≡ ∃ *l*. ((*pc*#*l*)@[*pc*'])∈(*paths* (*succsF* Π)) ∧ (*noAnn* (*pc*#*l*) (*an* Π))

1.6 Lemmas

lemma *idx-nth*:
a∈ *set l* ⟹ *l*!(*idx l a*) = *a*
by (*induct l*, *auto*)

lemma *idx-not-nth*:
a ∉ *set l* ⟹ *idx l a* = *length l*
by (*induct l*, *auto*)

lemma *paths-append*:
⟦ *ps*@(*p*#*ps*') ∈ *paths* (*succsF* Π); *ps* ≠ []; *ps*' ≠ [] ⟧ ⟹ (*p*#*ps*') ∈ (*paths* (*succsF* Π))
lemma *path-succsF*:
⟦ *l* ∈ *paths* (*succsF* Π); *k*+*1*<*length l* ⟧ ⟹ ∃ *B*. (*l*!(*k*+*1*),*B*) ∈ *set* (*succsF* Π (*l*!*k*))

lemma *less-chain-simp*:
$[\![\forall k.\ length\ l <= k+1 \vee idx\ L\ (l!k) < idx\ L\ (l!(k+1));\ 1 < length\ (l::'pos\ \ list)\ ]\!] \Longrightarrow idx\ L\ (hd\ l) < idx\ L\ (last\ l)$
lemma *path-length*:
$[\![l \in paths\ (succsF\ \Pi)\ ]\!] \Longrightarrow 1 < length\ l$
lemma *loop-has-back-jump*:
$[\![l \in paths\ (succsF\ \Pi);\ hd\ l = last\ l\ ]\!] \Longrightarrow \exists k.\ (k+1)<length\ l \wedge idx\ L\ (l!(k+1)) <= idx\ L\ (l!k)$
end