

A formalized programming language with speculative execution

Jamie Wright Andrei Popescu

February 6, 2026

Abstract

We present the formalization of a programming language whose operational semantics allows for the speculative execution of its statements. This type of semantics is relevant for discussing transient execution security vulnerabilities such as Spectre and Meltdown. An instantiation of Relative Security to this language is provided along with proofs of security and insecurity of selected programs from the Spectre benchmark.

Contents

1	A Simple Imperative Language	2
1.1	Arithmetic and Boolean Expressions	3
1.2	Commmands	3
1.3	Stores, States and Configurations	4
1.4	Evaluation of arithmetic and boolean expressions	5
2	Basic Semantics	6
2.1	Well-formed programs	6
2.2	Basic Semantics of Commands	7
2.3	State Transitions	9
2.3.1	Simplification Rules	10
2.3.2	Elimination Rules	13
2.4	Read locations	15
3	Normal Semantics	16
3.1	State Transitions	17
3.2	Elimination Rules	17
4	Misprediction and Speculative Semantics	18
4.1	Misprediction Oracle	18
4.2	Mispredicting Step	19

4.2.1	State Transitions	19
4.3	Speculative Semantics	21
4.3.1	State Transitions	23
4.3.2	Elimination Rules	27
5	Relative Security instantiation - Common Aspects	28
6	Relative Security Instance: Secret Memory	33
7	Relative Security Instance: Secret Memory Input	36
8	Disproof of Relative Security for fun1	39
8.1	Function definition and Boilerplate	40
8.2	Proof	48
8.2.1	Concrete leak	48
8.2.2	Auxillary lemmas for disproof	52
8.2.3	Disproof of fun1	53
9	Proof of Relative Security for fun2	55
9.1	Function definition and Boilerplate	56
9.2	Proof	63
10	Proof of Relative Security for fun3	69
10.1	Function definition and Boilerplate	69
10.2	Proof	77
11	Proof of Relative Security for fun4	84
11.1	Function definition and Boilerplate	84
11.2	Proof	93
12	Proof of Relative Security for fun5	103
12.1	Function definition and Boilerplate	103
12.2	Proof	114
13	Proof of Relative Security for fun6	121
13.1	Function definition and Boilerplate	121
13.2	Proof	133
14	Proof of Relative Security for fun2	141
14.1	Function definition and Boilerplate	141
14.2	Proof	149

1 A Simple Imperative Language

```
theory Language-Syntax imports Language-Prelims Relative-Security.Trivia begin
```

A Simple Imperative Language with arrays, inputs and outputs, and speculation fences, based off the syntax for IMP in Concrete Semantics [3]

Scalar variables are defined as strings, and so are the array variables

type-synonym *vname* = *string*
type-synonym *avname* = *string*

Since the Spectre benchmark examples reason about integer variables, we define our set of values to be integers

type-synonym *val* = *int*

We define our set of locations to be integers

type-synonym *loc* = *nat*

1.1 Arithmetic and Boolean Expressions

Arithmetic expressions can either be literals, variables or array variables (array variable name, index), or some operation on these. The arithmetic operators we capture in an expression are addition and multiplication. For boolean expressions we capture negation and conjunction, and the arithmetic comparison operator "less than" where equality of two arithmetic terms is later defined in terms of these constructors

datatype *aexp* = *N int* | *V vname* | *VA avname aexp* | *Plus aexp aexp* | *Times aexp aexp* |
Ite bexp aexp aexp | *Fun aexp aexp*
and *bexp* = *Bc bool* | *Not bexp* | *And bexp bexp* | *Less aexp aexp*

To enable reasoning about more subtle Spectre-like examples require the existence of trusted and untrusted I/O channels

datatype *trustStat* = *Trusted (T)* | *Untrusted (U)*

consts *func* :: *aexp* × *aexp* ⇒ *val*

A little syntax magic to write larger states compactly:

definition *null-state* (<>) **where**
null-state ≡ λ*x*. 0

syntax

-*State* :: *updbinds* => '*a*' (<->)

translations

-*State ms* == -*Update* <> *ms*

-*State (-updbinds b bs)* <= -*Update (-State b) bs*

1.2 Commmands

The language defined by this grammar capture standard basic mechanisms for manipulating scalar and array variables, and (un)conditional jumps, us-

ing Jump and IfJump, as control structures. It is also an I/O interactive language, accepting inputs on various input channels and producing outputs on various output channels. Most of the commands are standard, however there is an inclusion of Fences and Masking commands which are non-standard. The "Fence" command models the lfence instruction which prevents further speculative execution and is crucial in capturing key Spectre benchmark examples. The Mask command models Speculative Load Hardening (SLH), which masks variable values with respect to a given condition, contextually it can protect against leaks by masking values during misspeculation. It can be read as "M var I b T exp1 E exp2 == IF b THEN var = exp1 ELSE var = exp2"

```
datatype (discs-sels) com =
  | Start
  | Skip

  | getInput trustStat vname ((Input -/ -) [0, 61] 61)
  | Output trustStat aexp ((Output -/ -) [0, 61] 61)
  | Mask vname bexp aexp aexp (M -/ I -/ T -/ E - [1000, 61, 61, 61] 61)
  | Fence
  | Jump nat
  | Assign vname aexp (- ::= - [1000, 61] 61)
  | ArrAssign avname aexp aexp (- [-] ::= - [1000, 61] 61)
  | IfJump bexp nat nat ((IfJump -/ -/ -) [0, 0, 61] 61)
```

A predicate which determines whether or not a memory read occurs in an arithmetic expression

```
fun isReadMemory :: aexp ⇒ bool where
isReadMemory (N n) = False |
isReadMemory (V x) = False |
isReadMemory (VA a i) = True |
isReadMemory (Plus a1 a2) = (isReadMemory a1 ∨ isReadMemory a2)|
isReadMemory (Times a1 a2) = (isReadMemory a1 ∨ isReadMemory a2)
```

1.3 Stores, States and Configurations

Defining a variable store, array variable store and a heap. The variable store is as standard, mapping variable names to values. The array variable store maps array name, to a base address in the and the size of the array. The heap maps memory locations to values

```
datatype vstore = Vstore (vstore:vname ⇒ val)
datatype avstore = Avstore (avstore:avname ⇒ loc * nat)
datatype heap = Heap (hheap:loc ⇒ val)
```

A given value of an element in an array is assigned in the heap at location "array base+index". For example if the array "a1" has array base = 0, then the value a1[3] can be found at memory location 3 in the heap

definition *array-base* :: *avname* \Rightarrow *avstore* \Rightarrow *loc* **where**
array-base arr avst \equiv *case avst of (Avstore as) \Rightarrow fst (as arr)*

definition *array-bound* :: *avname* \Rightarrow *avstore* \Rightarrow *nat* **where**
array-bound arr avst \equiv *case avst of (Avstore as) \Rightarrow snd (as arr)*

definition *array-loc* :: *avname* \Rightarrow *nat* \Rightarrow *avstore* \Rightarrow *loc* **where**
array-loc arr i avst \equiv *array-base arr avst + i*

lemma *array-locBase*: *array-base arr avst = array-loc arr 0 avst*
 {*proof*}

A state consists of: (command, variable store, heap, next free location in the heap).

datatype *state* = *State (getVstore: vstore) (getAvstore: avstore) (getHeap: heap) (getFree: nat)*

fun *getHheap* **where** *getHheap (State vst avst h p) = hheap h*

A configuration for the normal semantics consists of: (command, state, the set of read memory locations so far).

type-synonym *pcounter* = *nat*

datatype *config* = *Config (pcOf: pcounter) (stateOf: state)*

fun *vstoreOf* **where** *vstoreOf (Config pc s) = vstore (getVstore s)*
fun *avstoreOf* **where** *avstoreOf (Config pc s) = avstore (getAvstore s)*
fun *heapOf* **where** *heapOf (Config pc s) = getHeap s*
fun *freeOf* **where** *freeOf (Config pc s) = getFree s*
fun *hheapOf* **where** *hheapOf (Config pc s) = getHheap s*

1.4 Evaluation of arithmetic and boolean expressions

A standard recursive function which evaluates a given expression

fun *aval* :: *aexp* \Rightarrow *state* \Rightarrow *val*
and *bval* :: *bexp* \Rightarrow *state* \Rightarrow *bool* **where**
aval (N n) s = n
 |
aval (V x) s = vstore (getVstore s) x
 |
aval (VA a i) s = getHheap s (array-loc a (nat(aval i s)) (getAvstore s))
 |
aval (Plus a1 a2) s = aval a1 s + aval a2 s
 |
*aval (Times a1 a2) s = aval a1 s * aval a2 s*
 |

```

aval (Ite b a1 a2) s = (if bval b s then aval a1 s else aval a2 s)
|
aval (Fun x y) s = func (x, y)
|
bval (Bc v) s = v
|
bval (Not b) s = (¬ bval b s)
|
bval (And b1 b2) s = (bval b1 s ∧ bval b2 s)
|
bval (Less a1 a2) s = (aval a1 s < aval a2 s)

```

An arithmetic equivalence of two terms as a boolean expression

definition $Eq :: aexp \Rightarrow aexp \Rightarrow bexp$ **where**
 $Eq\ a1\ a2 \equiv And\ (Not\ (Less\ a1\ a2))\ (Not\ (Less\ a2\ a1))$

lemma $Eq\text{-}verif: bval\ (Eq\ a1\ a2)\ s \longleftrightarrow aval\ a1\ s = aval\ a2\ s$
<proof>

fun $outOf :: com \Rightarrow state \Rightarrow val$ **where**
 $outOf\ c\ s = (case\ c\ of\ Output\ T\ aexp \Rightarrow aval\ aexp\ s \mid - \Rightarrow undefined)$

end

2 Basic Semantics

theory *Step-Basic*
imports *Language-Syntax*
begin

This theory introduces a standard semantics for the commands defined

2.1 Well-formed programs

A well-formed program is a nonempty list of commands where the head of the list is the "Start" command

type-synonym $prog = com\ list$

locale $Prog =$
fixes $prog :: prog$
assumes
 $wf\text{-}prog: prog \neq [] \wedge hd\ prog = Start$
begin

This is the program counter signifying the end of the program:

definition $endPC \equiv length\ prog$

And some sanity checks for a well formed program...

lemma $lenth-prog-gt-0$: $length\ prog > 0$
 $\langle proof \rangle$

lemma $lenth-prog-not-0$: $length\ prog \neq 0$
 $\langle proof \rangle$

lemma $endPC-gt-0$: $endPC > 0$
 $\langle proof \rangle$

lemma $endPC-not-0$: $endPC \neq 0$
 $\langle proof \rangle$

lemma $hd-prog-Start$: $hd\ prog = Start$
 $\langle proof \rangle$

lemma $prog-0$: $prog ! 0 = Start$
 $\langle proof \rangle$

2.2 Basic Semantics of Commands

The basic small step semantics of the language, parameterised by a fixed program. The semantics operate on input streams and memories which are consumed and updated while the program counter moves through the list of commands. This emulates standard (and expected) execution of the commands defined. Since no speculation is captured in this basic semantics, the Fence command the same as SKIP

inductive

$stepB :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow bool$ (**infix** $\rightarrow B\ 55$)

where

Seq-Start-Skip-Fence:

$pc < endPC \Longrightarrow prog!pc \in \{Start, Skip, Fence\} \Longrightarrow$
 $(Config\ pc\ s, ibT, ibUT) \rightarrow B (Config\ (Suc\ pc)\ s, ibT, ibUT)$

|

Assign:

$pc < endPC \Longrightarrow prog!pc = (x ::= a) \Longrightarrow$
 $s = State\ (Vstore\ vs)\ avst\ h\ p \Longrightarrow$
 $(Config\ pc\ s, ibT, ibUT) \rightarrow B$
 $(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := aval\ a\ s))))\ avst\ h\ p), ibT, ibUT)$

|

ArrAssign:

$pc < endPC \Longrightarrow prog!pc = (arr[index] ::= a) \Longrightarrow$
 $v = aval\ index\ s \Longrightarrow w = aval\ a\ s \Longrightarrow$

$$\begin{aligned}
& 0 \leq v \implies v < \text{int } (\text{array-bound } \text{arr } \text{avst}) \implies \\
& l = \text{array-loc } \text{arr } (\text{nat } v) \text{ avst} \implies \\
& s = \text{State } \text{vst } \text{avst } (\text{Heap } h) p \\
& \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \\
& \rightarrow B \\
& (\text{Config } (\text{Suc } pc) (\text{State } \text{vst } \text{avst } (\text{Heap } (h(l := w)))) p), \text{ibT}, \text{ibUT}) \\
& | \\
& \text{getTrustedInput:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{Input } T \ x \implies \\
& (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{avst } h \ p), \text{LCons } i \ \text{ibT}, \text{ibUT}) \\
& \rightarrow B \\
& (\text{Config } (\text{Suc } pc) (\text{State } (\text{Vstore } (vs(x := i))) \text{avst } h \ p), \text{ibT}, \text{ibUT}) \\
& | \\
& \text{getUntrustedInput:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{Input } U \ x \implies \\
& (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{avst } h \ p), \text{ibT}, \text{LCons } i \ \text{ibUT}) \\
& \rightarrow B \\
& (\text{Config } (\text{Suc } pc) (\text{State } (\text{Vstore } (vs(x := i))) \text{avst } h \ p), \text{ibT}, \text{ibUT}) \\
& | \\
& \text{Output:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{Output } t \ \text{aexp} \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \\
& \rightarrow B \\
& (\text{Config } (\text{Suc } pc) \ s, \text{ibT}, \text{ibUT}) \\
& | \\
& \text{Jump:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{Jump } pc1 \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \rightarrow B (\text{Config } pc1 \ s, \text{ibT}, \text{ibUT}) \\
& | \\
& \text{IfTrue:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{IfJump } b \ pc1 \ pc2 \implies \\
& \text{bval } b \ s \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \rightarrow B (\text{Config } pc1 \ s, \text{ibT}, \text{ibUT}) \\
& | \\
& \text{IfFalse:} \\
& pc < \text{endPC} \implies \text{prog!pc} = \text{IfJump } b \ pc1 \ pc2 \implies \\
& \neg \text{bval } b \ s \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \rightarrow B (\text{Config } pc2 \ s, \text{ibT}, \text{ibUT}) \\
& | \\
& \text{MaskTrue:} \\
& pc < \text{endPC} \implies \text{prog!pc} = (\text{M } x \ I \ b \ T \ a1 \ E \ a2 \) \implies \\
& \text{bval } b \ s \implies \\
& s = \text{State } (\text{Vstore } vs) \text{avst } h \ p \implies \\
& (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \\
& \rightarrow B \\
& (\text{Config } (\text{Suc } pc) (\text{State } (\text{Vstore } (vs(x := \text{aval } a1 \ s)))) \text{avst } h \ p), \text{ibT}, \text{ibUT}) \\
& | \\
& \text{MaskFalse:}
\end{aligned}$$

lemma *stepB-determ*:

$cfg\text{-}ib \rightarrow B\ cfg\text{-}ib' \implies cfg\text{-}ib \rightarrow B\ cfg\text{-}ib'' \implies cfg\text{-}ib'' = cfg\text{-}ib'$

<proof>

definition *nextB* :: $config \times val\ list \times val\ list \Rightarrow config \times val\ list \times val\ list$

where

$nextB\ cfg\text{-}ib \equiv SOME\ cfg'\text{-}ib'.\ cfg\text{-}ib \rightarrow B\ cfg'\text{-}ib'$

lemma *nextB-stepB*: $\neg\ finalB\ cfg\text{-}ib \implies cfg\text{-}ib \rightarrow B\ (nextB\ cfg\text{-}ib)$

<proof>

lemma *stepB-nextB*: $cfg\text{-}ib \rightarrow B\ cfg'\text{-}ib' \implies cfg'\text{-}ib' = nextB\ cfg\text{-}ib$

<proof>

lemma *nextB-iff-stepB*: $\neg\ finalB\ cfg\text{-}ib \implies nextB\ cfg\text{-}ib = cfg'\text{-}ib' \iff cfg\text{-}ib \rightarrow B\ cfg'\text{-}ib'$

<proof>

lemma *stepB-iff-nextB*: $cfg\text{-}ib \rightarrow B\ cfg'\text{-}ib' \iff \neg\ finalB\ cfg\text{-}ib \wedge nextB\ cfg\text{-}ib = cfg'\text{-}ib'$

<proof>

2.3.1 Simplification Rules

Sufficient conditions for a given command to "execute" transit to the next state

lemma *nextB-Start-Skip-Fence[simp]*:

$pc < endPC \implies prog!pc \in \{Start, Skip, Fence\} \implies$

$nextB\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ (Suc\ pc)\ s,\ ibT,\ ibUT)$

<proof>

lemma *nextB-Assign[simp]*:

$pc < endPC \implies prog!pc = (x ::= a) \implies$

$s = State\ (Vstore\ vs)\ avst\ h\ p \implies$

$nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$

$=$

$(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x ::= aval\ a\ s))))\ avst\ h\ p),$

$ibT,\ ibUT)$

<proof>

lemma *nextB-ArrAssign[simp]*:

$pc < endPC \implies prog!pc = (arr[index] ::= a) \implies$

$ls' = readLocs\ a\ vst\ avst\ (Heap\ h) \implies$

$v = aval\ index\ s \implies w = aval\ a\ s \implies$

$0 \leq v \implies v < int\ (array\ bound\ arr\ avst) \implies$

$l = array\ loc\ arr\ (nat\ v)\ avst \implies$

$s = State\ vst\ avst\ (Heap\ h)\ p$

\implies

$nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$

=
 (Config (Suc pc) (State vst avst (Heap (h(l := w))) p), ibT, ibUT)
 ⟨proof⟩

lemma nextB-getTrustedInput[simp]:
 $pc < endPC \implies prog!pc = (Input\ T\ x) \implies$
 $nextB\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ LCons\ i\ ibT,\ ibUT)$
 =
 (Config (Suc pc) (State (Vstore (vs(x := i))) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextB-getUntrustedInput[simp]:
 $pc < endPC \implies prog!pc = (Input\ U\ x) \implies$
 $nextB\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ LCons\ i\ ibUT)$
 =
 (Config (Suc pc) (State (Vstore (vs(x := i))) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextB-getTrustedInput'[simp]:
 $pc < endPC \implies prog!pc = Input\ T\ x \implies$
 $ibT \neq LNil \implies$
 $nextB\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 =
 (Config (Suc pc) (State (Vstore (vs(x := lhd ibT))) avst h p), ltl ibT, ibUT)
 ⟨proof⟩

lemma nextB-getUntrustedInput'[simp]:
 $pc < endPC \implies prog!pc = Input\ U\ x \implies$
 $ibUT \neq LNil \implies$
 $nextB\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 =
 (Config (Suc pc) (State (Vstore (vs(x := lhd ibUT))) avst h p), ibT, ltl ibUT)
 ⟨proof⟩

lemma nextB-Output[simp]:
 $pc < endPC \implies prog!pc = Output\ t\ aexp \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
 =
 (Config (Suc pc) s, ibT, ibUT)
 ⟨proof⟩

lemma nextB-Jump[simp]:
 $pc < endPC \implies prog!pc = Jump\ pc1 \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc1\ s,\ ibT,\ ibUT)$
 ⟨proof⟩

lemma nextB-IfTrue[simp]:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $bval\ b\ s \implies$

$nextB (Config\ pc\ s, ibT, ibUT) = (Config\ pc1\ s, ibT, ibUT)$
 ⟨proof⟩

lemma *nextB-IfFalse*[simp]:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $\neg bval\ b\ s \implies$
 $nextB (Config\ pc\ s, ibT, ibUT) = (Config\ pc2\ s, ibT, ibUT)$
 ⟨proof⟩

lemma *nextB-MaskTrue*[simp]:
 $pc < endPC \implies prog!pc = (M\ x\ I\ b\ T\ a1\ E\ a2) \implies$
 $bval\ b\ s \implies$
 $s = State\ (Vstore\ vs)\ avst\ h\ p \implies$
 $nextB (Config\ pc\ s, ibT, ibUT)$
 $=$
 $(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := aval\ a1\ s))))\ avst\ h\ p),$
 $ibT, ibUT)$
 ⟨proof⟩

lemma *nextB-MaskFalse*[simp]:
 $pc < endPC \implies prog!pc = (M\ x\ I\ b\ T\ a1\ E\ a2) \implies$
 $\neg bval\ b\ s \implies$
 $s = State\ (Vstore\ vs)\ avst\ h\ p \implies$
 $nextB (Config\ pc\ s, ibT, ibUT)$
 $=$
 $(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := aval\ a2\ s))))\ avst\ h\ p),$
 $ibT, ibUT)$
 ⟨proof⟩

lemma *finalB-endPC*: $pcOf\ cfg = endPC \implies finalB (cfg, ibT, ibUT)$
 ⟨proof⟩

lemma *stepB-endPC*: $pcOf\ cfg = endPC \implies \neg (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT')$
 ⟨proof⟩

lemma *stepB-imp-le-endPC*: **assumes** $(cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT')$
shows $pcOf\ cfg < endPC$
 ⟨proof⟩

lemma *stepB-0*: $(Config\ 0\ s, ibT, ibUT) \rightarrow B (Config\ 1\ s, ibT, ibUT)$
 ⟨proof⟩

2.3.2 Elimination Rules

In the unwinding proofs of relative security it is often the case that two traces will progress in lockstep, when doing so we wish to preserve/update invariants of the current state. The following are some useful elimination rules to help simplify reasoning

lemma *stepB-Seq-Start-Skip-FenceE*:

assumes $\langle \text{cfg}, \text{ibT}, \text{ibUT} \rangle \rightarrow B \langle \text{cfg}', \text{ibT}', \text{ibUT}' \rangle$
and $\langle \text{cfg} = (\text{Config } pc \text{ (State (Vstore } vs) \text{ avst } h \text{ p})) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' \text{ (State (Vstore } vs') \text{ avst}' h' \text{ p}')) \rangle$
and $\langle \text{prog!pc} \in \{\text{Start}, \text{Skip}, \text{Fence}\} \rangle$
shows $\langle vs' = vs \wedge \text{ibT} = \text{ibT}' \wedge \text{ibUT} = \text{ibUT}' \wedge$
 $pc' = \text{Suc } pc \wedge \text{avst}' = \text{avst} \wedge h' = h \wedge$
 $p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-AssignE*:

assumes $\langle \text{cfg}, \text{ibT}, \text{ibUT} \rangle \rightarrow B \langle \text{cfg}', \text{ibT}', \text{ibUT}' \rangle$
and $\langle \text{cfg} = (\text{Config } pc \text{ (State (Vstore } vs) \text{ avst } h \text{ p})) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' \text{ (State (Vstore } vs') \text{ avst}' h' \text{ p}')) \rangle$
and $\langle \text{prog!pc} = (x ::= a) \rangle$
shows $\langle vs' = (vs(x ::= \text{aval } a \text{ (stateOf } \text{cfg}))) \wedge$
 $\text{ibT} = \text{ibT}' \wedge \text{ibUT} = \text{ibUT}' \wedge pc' = \text{Suc } pc \wedge$
 $\text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-getTrustedInputE*:

assumes $\langle \text{cfg}, \text{ibT}, \text{ibUT} \rangle \rightarrow B \langle \text{cfg}', \text{ibT}', \text{ibUT}' \rangle$
and $\langle \text{cfg} = (\text{Config } pc \text{ (State (Vstore } vs) \text{ avst } h \text{ p})) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' \text{ (State (Vstore } vs') \text{ avst}' h' \text{ p}')) \rangle$
and $\langle \text{prog!pc} = \text{Input } T \text{ } x \rangle$
shows $\langle vs' = (vs(x := \text{lhd } \text{ibT})) \wedge$
 $\text{ibT}' = \text{lhl } \text{ibT} \wedge \text{ibUT} = \text{ibUT}' \wedge pc' = \text{Suc } pc \wedge$
 $\text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-getUntrustedInputE*:

assumes $\langle \text{cfg}, \text{ibT}, \text{ibUT} \rangle \rightarrow B \langle \text{cfg}', \text{ibT}', \text{ibUT}' \rangle$
and $\langle \text{cfg} = (\text{Config } pc \text{ (State (Vstore } vs) \text{ avst } h \text{ p})) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' \text{ (State (Vstore } vs') \text{ avst}' h' \text{ p}')) \rangle$
and $\langle \text{prog!pc} = \text{Input } U \text{ } x \rangle$
shows $\langle vs' = (vs(x := \text{lhd } \text{ibUT})) \wedge$
 $\text{ibT}' = \text{ibT} \wedge \text{ibUT}' = \text{lhl } \text{ibUT} \wedge pc' = \text{Suc } pc \wedge$
 $\text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-OutputE*:

assumes $\langle \text{cfg}, \text{ibT}, \text{ibUT} \rangle \rightarrow B \langle \text{cfg}', \text{ibT}', \text{ibUT}' \rangle$
and $\langle \text{cfg} = (\text{Config } pc \text{ (State (Vstore } vs) \text{ avst } h \text{ p})) \rangle$

and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{Output } t \text{ aexp} \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = \text{Suc } pc \wedge \text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-JumpE*:

assumes $\langle (\text{cfg}, ibT, ibUT) \rightarrow B (\text{cfg}', ibT', ibUT') \rangle$
and $\langle \text{cfg} = (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{ avst } h p)) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{Jump } pc1 \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = pc1 \wedge \text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-IfTrueE*:

assumes $\langle (\text{cfg}, ibT, ibUT) \rightarrow B (\text{cfg}', ibT', ibUT') \rangle$
and $\langle \text{cfg} = (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{ avst } h p)) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{IfJump } b \text{ pc1 } pc2 \rangle$ **and** $\langle \text{bval } b (\text{stateOf } \text{cfg}) \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = pc1 \wedge \text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-IfFalseE*:

assumes $\langle (\text{cfg}, ibT, ibUT) \rightarrow B (\text{cfg}', ibT', ibUT') \rangle$
and $\langle \text{cfg} = (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{ avst } h p)) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{IfJump } b \text{ pc1 } pc2 \rangle$ **and** $\langle \neg \text{bval } b (\text{stateOf } \text{cfg}) \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = pc2 \wedge \text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-MaskTrueE*:

assumes $\langle (\text{cfg}, ibT, ibUT) \rightarrow B (\text{cfg}', ibT', ibUT') \rangle$
and $\langle \text{cfg} = (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{ avst } h p)) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{M } x \text{ I } b \text{ T } a1 \text{ E } a2 \rangle$ **and** $\langle \text{bval } b (\text{stateOf } \text{cfg}) \rangle$
shows $\langle vs' = (vs(x := \text{aval } a1 (\text{stateOf } \text{cfg}))) \wedge$
 $ibT = ibT' \wedge ibUT = ibUT' \wedge pc' = \text{Suc } pc \wedge$
 $\text{avst}' = \text{avst} \wedge h' = h \wedge p' = p \rangle$
 $\langle \text{proof} \rangle$

lemma *stepB-MaskFalseE*:

assumes $\langle (\text{cfg}, ibT, ibUT) \rightarrow B (\text{cfg}', ibT', ibUT') \rangle$
and $\langle \text{cfg} = (\text{Config } pc (\text{State } (\text{Vstore } vs) \text{ avst } h p)) \rangle$
and $\langle \text{cfg}' = (\text{Config } pc' (\text{State } (\text{Vstore } vs') \text{ avst}' h' p')) \rangle$
and $\langle \text{prog!pc} = \text{M } x \text{ I } b \text{ T } a1 \text{ E } a2 \rangle$ **and** $\langle \neg \text{bval } b (\text{stateOf } \text{cfg}) \rangle$
shows $\langle vs' = (vs(x := \text{aval } a2 (\text{stateOf } \text{cfg}))) \wedge$

$$\begin{aligned}
ibT &= ibT' \wedge ibUT = ibUT' \wedge pc' = Suc\ pc \wedge \\
avst' &= avst \wedge h' = h \wedge p' = p
\end{aligned}$$

<proof>

end

2.4 Read locations

For modeling Spectre-like vulnerabilities, we record memory reads (as in [1]), i.e., accessed for reading during the execution. We let $readLocs(pc,u)$ be the (possibly empty) set of locations that are read when executing the current command c - computed from all sub-expressions of the form $a[e]$. i.e. array reads. For example, if c is the assignment $"x = a [b[3]]"$, then $readLocs$ returns two locations: counting from 0, the 3rd location of b and the $b[3]$ 'th location of a .

```

fun readLocsA :: aexp ⇒ state ⇒ loc set and
readLocsB :: bexp ⇒ state ⇒ loc set where
readLocsA (N n) s = {}
|
readLocsA (V x) s = {}
|
readLocsA (VA arr index) s =
  insert (array-loc arr (nat (aval index s)) (getAvstore s))
    (readLocsA index s)
|
readLocsA (Plus a1 a2) s = readLocsA a1 s ∪ readLocsA a2 s
|
readLocsA (Times a1 a2) s = readLocsA a1 s ∪ readLocsA a2 s
|
readLocsA (Ite b a1 a2) s = readLocsB b s ∪ readLocsA a1 s ∪ readLocsA a2 s
|
readLocsA (Fun a b) s = {}
|
readLocsB (Bc c) s = {}
|
readLocsB (Not b) s = readLocsB b s
|
readLocsB (And b1 b2) s = readLocsB b1 s ∪ readLocsB b2 s
|
readLocsB (Less a1 a2) s = readLocsA a1 s ∪ readLocsA a2 s

```

```

fun readLocsC :: com ⇒ state ⇒ loc set where
readLocsC (x ::= a) s = readLocsA a s
|
readLocsC (arr[index] ::= a) s = readLocsA (VA arr index) s ∪ readLocsA a s
|
readLocsC (Output t a) s = readLocsA a s

```

```

|
readLocsC (IfJump b n1 n2) s = readLocsB b s
|
readLocsC (M x I b T a1 E a2) s = readLocsB b s ∪ (if (bval b s) then readLocsA
a1 s
                                     else readLocsA a2 s)
|
readLocsC - - = {}

```

```

context Prog
begin

```

```

definition readLocs cfg ≡ readLocsC (prog!(pcOf cfg)) (stateOf cfg)

```

```

end

```

```

end

```

3 Normal Semantics

This theory augments the basic semantics to include a set of read locations which is a simple representation of a cache

The normal semantics is defined by a single rule which involves the basic semantics, extended to accumulate the read locations, which accounts for cache side-channels

```

theory Step-Normal
imports Step-Basic
begin

```

```

context Prog
begin

```

```

fun stepN :: config × val llist × val llist × loc set ⇒ config × val llist × val llist
× loc set ⇒ bool (infix ‹→N› 55)

```

```

where

```

```

(cfg, ibT, ibUT, ls) →N (cfg', ibT', ibUT', ls') =
((cfg, ibT, ibUT) →B (cfg', ibT', ibUT')) ∧ ls' = ls ∪ readLocs cfg)

```

```

abbreviation

```

```

stepsN :: config × val llist × val llist × loc set ⇒ config × val llist × val llist ×
loc set ⇒ bool (infix ‹→N*› 55)

```

```

where x →N* y == star stepN x y

```

definition $finalN = final\ stepN$
lemmas $finalN-defs = final-def\ finalN-def$

lemma $finalN-iff-finalB[simp]$:
 $finalN\ (cfg,\ ibT,\ ibUT,\ ls) \longleftrightarrow finalB\ (cfg,\ ibT,\ ibUT)$
 $\langle proof \rangle$

3.1 State Transitions

fun $nextN :: config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist$
 $\times loc\ set$ **where**
 $nextN\ (cfg,\ ibT,\ ibUT,\ ls) = (case\ nextB\ (cfg,ibT,ibUT)\ of\ (cfg',ibT',ibUT') \Rightarrow$
 $(cfg',ibT',ibUT',ls \cup readLocs\ cfg))$

lemma $nextN-stepN$: $\neg finalN\ cfg-ib-ls \Longrightarrow cfg-ib-ls \rightarrow N\ (nextN\ cfg-ib-ls)$
 $\langle proof \rangle$

lemma $stepN-nextN$: $cfg-ib-ls \rightarrow N\ cfg'-ib'-ls' \Longrightarrow cfg'-ib'-ls' = nextN\ cfg-ib-ls$
 $\langle proof \rangle$

lemma $nextN-iff-stepN$:
 $\neg finalN\ cfg-ib-ls \Longrightarrow nextN\ cfg-ib-ls = cfg'-ib'-ls' \longleftrightarrow cfg-ib-ls \rightarrow N\ cfg'-ib'-ls'$
 $\langle proof \rangle$

lemma $stepN-iff-nextN$: $cfg-ib-ls \rightarrow N\ cfg'-ib'-ls' \longleftrightarrow \neg finalN\ cfg-ib-ls \wedge nextN\$
 $cfg-ib-ls = cfg'-ib'-ls'$
 $\langle proof \rangle$

lemma $finalN-endPC$: $pcOf\ cfg = endPC \Longrightarrow finalN\ (cfg,ibT,ibUT)$
 $\langle proof \rangle$

lemma $stepN-endPC$: $pcOf\ cfg = endPC \Longrightarrow \neg (cfg,ibT,ibUT) \rightarrow N\ (cfg',ibT',ibUT')$
 $\langle proof \rangle$

lemma $stepN-0$: $(Config\ 0\ s,\ ibT,\ ibUT,\ ls) \rightarrow N\ (Config\ 1\ s,\ ibT,\ ibUT,\ ls)$
 $\langle proof \rangle$

lemma $finalB-eq-finalN$: $finalB\ (cfg,\ ibT,ibUT) \longleftrightarrow (\forall\ ls.\ finalN\ (cfg,\ ibT,ibUT,$
 $ls))$
 $\langle proof \rangle$

3.2 Elimination Rules

lemma $stepN-Assign2E$:
assumes $\langle (cfg1,\ ibT1,ibUT1,\ ls1) \rightarrow N\ (cfg1',\ ibT1',ibUT1',\ ls1') \rangle$

```

and ⟨(cfg2, ibT2,ibUT2, ls2) →N (cfg2', ibT2',ibUT2', ls2')⟩
and ⟨cfg1 = (Config pc1 (State (Vstore vs1) avst1 h1 p1))⟩ and ⟨cfg1' =
(Config pc1' (State (Vstore vs1') avst1' h1' p1'))⟩
and ⟨cfg2 = (Config pc2 (State (Vstore vs2) avst2 h2 p2))⟩ and ⟨cfg2' =
(Config pc2' (State (Vstore vs2') avst2' h2' p2'))⟩
and ⟨prog!pc1 = (x ::= a)⟩ and ⟨pcOf cfg1 = pcOf cfg2⟩
shows ⟨vs1' = (vs1(x := aval a (stateOf cfg1))) ∧ ibT1 = ibT1' ∧ ibUT1 =
ibUT1' ∧
vs2' = (vs2(x := aval a (stateOf cfg2))) ∧ ibT2 = ibT2' ∧ ibUT2 =
ibUT2' ∧
pc1' = Suc pc1 ∧ pc2' = Suc pc2 ∧ ls2' = ls2 ∪ readLocs cfg2 ∧
avst1' = avst1 ∧ avst2' = avst2 ∧ ls1' = ls1 ∪ readLocs cfg1⟩
⟨proof⟩

```

lemma *stepN-Seq-Start-Skip-Fence2E*:

```

assumes ⟨(cfg1, ibT1,ibUT1, ls1) →N (cfg1', ibT1',ibUT1', ls1')⟩
and ⟨(cfg2, ibT2,ibUT2, ls2) →N (cfg2', ibT2',ibUT2', ls2')⟩
and ⟨cfg1 = (Config pc1 (State (Vstore vs1) avst1 h1 p1))⟩ and ⟨cfg1' =
(Config pc1' (State (Vstore vs1') avst1' h1' p1'))⟩
and ⟨cfg2 = (Config pc2 (State (Vstore vs2) avst2 h2 p2))⟩ and ⟨cfg2' =
(Config pc2' (State (Vstore vs2') avst2' h2' p2'))⟩
and ⟨prog!pc1 ∈ {Start, Skip, Fence}⟩ and ⟨pcOf cfg1 = pcOf cfg2⟩
shows ⟨vs1' = vs1 ∧ vs2' = vs2 ∧
pc1' = Suc pc1 ∧ pc2' = Suc pc2 ∧
avst1' = avst1 ∧ avst2' = avst2 ∧
ls2' = ls2 ∧ ls1' = ls1⟩
⟨proof⟩

```

end

end

4 Misprediction and Speculative Semantics

This theory formalizes an optimized speculative semantics, which allows for a characterization of the Spectre vulnerability, this work is inspired and based off the speculative semantics introduced by Cheang et al. [1]

```

theory Step-Spec
imports Step-Basic
begin

```

4.1 Misprediction Oracle

The speculative semantics is parameterised by a misprediction oracle. This consists of a predictor state:

typedecl *predState*

Along with predicates "mispred" (which decides when a misprediction occurs), "resolve" (which decides for when a speculation is resolved)

Both depend on the predictor state (which evolves via the update function) and the program counters of nested speculation

```

locale Prog-Mispred =
  Prog prog
for prog :: com list
  +
fixes mispred :: predState ⇒ pcounter list ⇒ bool
and resolve :: predState ⇒ pcounter list ⇒ bool
and update :: predState ⇒ pcounter list ⇒ predState
begin

```

4.2 Mispredicting Step

stepM simply goes the other way than stepB at branches

inductive

stepM :: config × val llist × val llist ⇒ config × val llist × val llist ⇒ bool (**infix** \rightarrow_M 55)

where

IfTrue[intro]:

$$pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$$

$$bval\ b\ s \implies$$

$$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_M (Config\ pc2\ s,\ ibT,\ ibUT)$$

|

IfFalse[intro]:

$$pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$$

$$\neg\ bval\ b\ s \implies$$

$$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_M (Config\ pc1\ s,\ ibT,\ ibUT)$$

4.2.1 State Transitions

definition *finalM = final stepM*

lemma *finalM-iff-aux:*

$$pc < endPC \wedge is-IfJump\ (prog!pc)$$

$$\longleftrightarrow$$

$$(\exists\ cfg'. (Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_M\ cfg')$$

<proof>

lemma *finalM-iff:*

$$finalM\ (Config\ pc\ (State\ vst\ avst\ h\ p),\ ibT,\ ibUT)$$

$$\longleftrightarrow$$

$$(pc \geq endPC \vee \neg\ is-IfJump\ (prog!pc))$$

<proof>

lemma *finalB-imp-finalM*:
 $finalB (cfg, ibT, ibUT) \implies finalM (cfg, ibT, ibUT)$
 $\langle proof \rangle$

lemma *not-finalM-imp-not-finalB*:
 $\neg finalM (cfg, ibT, ibUT) \implies \neg finalB (cfg, ibT, ibUT)$
 $\langle proof \rangle$

lemma *stepM-determ*:
 $cfg\text{-}ib \rightarrow M cfg\text{-}ib' \implies cfg\text{-}ib \rightarrow M cfg\text{-}ib'' \implies cfg\text{-}ib'' = cfg\text{-}ib'$
 $\langle proof \rangle$

definition *nextM* :: $config \times val\ list \times val\ list \Rightarrow config \times val\ list \times val\ list$
where
 $nextM\ cfg\text{-}ib \equiv SOME\ cfg'\text{-}ib'.\ cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib'$

lemma *nextM-stepM*: $\neg finalM\ cfg\text{-}ib \implies cfg\text{-}ib \rightarrow M (nextM\ cfg\text{-}ib)$
 $\langle proof \rangle$

lemma *stepM-nextM*: $cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib' \implies cfg'\text{-}ib' = nextM\ cfg\text{-}ib$
 $\langle proof \rangle$

lemma *nextM-iff-stepM*: $\neg finalM\ cfg\text{-}ib \implies nextM\ cfg\text{-}ib = cfg'\text{-}ib' \iff cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib'$
 $\langle proof \rangle$

lemma *stepM-iff-nextM*: $cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib' \iff \neg finalM\ cfg\text{-}ib \wedge nextM\ cfg\text{-}ib = cfg'\text{-}ib'$
 $\langle proof \rangle$

lemma *nextM-IfTrue[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $\neg bval\ b\ s \implies$
 $nextM (Config\ pc\ s, ibT, ibUT) = (Config\ pc1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma *nextM-IfFalse[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $bval\ b\ s \implies$
 $nextM (Config\ pc\ s, ibT, ibUT) = (Config\ pc2\ s, ibT, ibUT)$
 $\langle proof \rangle$

end

4.3 Speculative Semantics

A "speculative" configuration is a quadruple consisting of:

- The predictor's state
- The nonspeculative configuration (at level 0 so to speak)
- The list of speculative configurations (modelling nested speculation, levels 1 to n, from left to right: so the last in this list is at the current speculation level, n)
- The list of inputs in the input buffer

We think of cfgs as a stack of configurations, one for each speculation level in a nested speculative execution. At level 0 (empty list) we have the configuration for normal, non-speculative execution. At each moment, only the top of the configuration stack, "hd cfgs" is active.

type-synonym $configS = predState \times config \times config\ list \times val\ llist \times val\ llist \times loc\ set$

context *Prog-Mispred*
begin

The speculative semantics is more involved than both the normal and basic semantics, so a short description of each rule is provided:

- `Non_spec_normal`: when we are either not mispredicting or not at a branch and there is no current speculation, i.e. normal execution
- `Nonspec_mispred`: when we are mispredicting and at a branch, speculation occurs down the wrong branch, i.e. branch misprediction
- `Spec_normal`: when we are either not mispredicting or not at a branch BUT there is speculation, i.e. standard speculative execution
- `Spec_mispred`: when we are mispredicting and at a branch, AND also speculating... speculation occurs down the wrong branch, and we go to another speculation level i.e. nested speculative execution
- `Spec_Fence`: when there is current speculation and a Fence is hit, all speculation resolves
- `Spec Resolve`: If the resolve predicate is true, resolution occurs for one speculation level. In contrast to Fences, resolve does not necessarily kill all speculation levels, but allows resolution one level at a time

inductive

stepS :: *configS* ⇒ *configS* ⇒ *bool* (**infix** →*S* 55)

where

nonspec-normal:

cfgs = [] ⇒
 ¬ *is-IfJump* (*prog!*(*pcOf* *cfg*)) ∨ ¬ *mispred* *pstate* [*pcOf* *cfg*] ⇒
pstate' = *pstate* ⇒
 ¬ *finalB* (*cfg*, *ibT*, *ibUT*) ⇒ (*cfg'*, *ibT'*, *ibUT'*) = *nextB* (*cfg*, *ibT*, *ibUT*) ⇒
cfgs' = [] ⇒
ls' = *ls* ∪ *readLocs* *cfg*
 ⇒
 (*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) →*S* (*pstate'*, *cfg'*, *cfgs'*, *ibT'*, *ibUT'*, *ls'*)

|

nonspec-mispred:

cfgs = [] ⇒
is-IfJump (*prog!*(*pcOf* *cfg*)) ⇒ *mispred* *pstate* [*pcOf* *cfg*] ⇒
pstate' = *update* *pstate* [*pcOf* *cfg*] ⇒
 ¬ *finalM* (*cfg*, *ibT*, *ibUT*) ⇒ (*cfg'*, *ibT'*, *ibUT'*) = *nextB* (*cfg*, *ibT*, *ibUT*) ⇒
 (*cfg1'*, *ibT1'*, *ibUT1'*) = *nextM* (*cfg*, *ibT*, *ibUT*) ⇒
cfgs' = [*cfg1* ^] ⇒
ls' = *ls* ∪ *readLocs* *cfg*
 ⇒
 (*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) →*S* (*pstate'*, *cfg'*, *cfgs'*, *ibT'*, *ibUT'*, *ls'*)

|

spec-normal:

cfgs ≠ [] ⇒
 ¬ *resolve* *pstate* (*pcOf* *cfg* # *map* *pcOf* *cfgs*) ⇒
 ¬ *is-IfJump* (*prog!*(*pcOf* (*last* *cfgs*))) ∨ ¬ *mispred* *pstate* (*pcOf* *cfg* # *map* *pcOf* *cfgs*) ⇒
prog!(*pcOf* (*last* *cfgs*)) ≠ *Fence* ⇒
pstate' = *pstate* ⇒
 ¬ *is-getInput* (*prog!*(*pcOf* (*last* *cfgs*))) ⇒
 ¬ *is-Output* (*prog!*(*pcOf* (*last* *cfgs*))) ⇒
 ¬ *finalB* (*last* *cfgs*, *ibT*, *ibUT*) ⇒ (*cfg1'*, *ibT'*, *ibUT'*) = *nextB* (*last* *cfgs*, *ibT*, *ibUT*) ⇒
cfg' = *cfg* ⇒ *cfgs'* = *butlast* *cfgs* @ [*cfg1* ^] ⇒
ls' = *ls* ∪ *readLocs* (*last* *cfgs*)
 ⇒
 (*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) →*S* (*pstate'*, *cfg'*, *cfgs'*, *ibT'*, *ibUT'*, *ls'*)

|

spec-mispred:

cfgs ≠ [] ⇒
 ¬ *resolve* *pstate* (*pcOf* *cfg* # *map* *pcOf* *cfgs*) ⇒
is-IfJump (*prog!*(*pcOf* (*last* *cfgs*))) ⇒ *mispred* *pstate* (*pcOf* *cfg* # *map* *pcOf* *cfgs*) ⇒
 ⇒
pstate' = *update* *pstate* (*pcOf* *cfg* # *map* *pcOf* *cfgs*) ⇒
 ¬ *finalM* (*last* *cfgs*, *ibT*, *ibUT*) ⇒
 (*lcfg'*, *ibT'*, *ibUT'*) = *nextB* (*last* *cfgs*, *ibT*, *ibUT*) ⇒ (*cfg1'*, *ibT1'*, *ibUT1'*) = *nextM* (*last* *cfgs*, *ibT*, *ibUT*) ⇒

$$\begin{aligned}
& cfg' = cfg \implies cfgs' = \text{butlast } cfgs @ [lcfg'] @ [cfg1'] \implies \\
& ls' = ls \cup \text{readLocs } (\text{last } cfgs) \\
& \implies \\
& (pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls') \\
& | \\
& \text{spec-Fence:} \\
& cfgs \neq [] \implies \\
& \neg \text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfgs) \implies \\
& \text{prog!}(pcOf\ (\text{last } cfgs)) = \text{Fence} \implies \\
& pstate' = pstate \implies cfg' = cfg \implies cfs' = [] \implies \\
& ibT = ibT' \implies ibUT = ibUT' \implies ls' = ls \\
& \implies \\
& (pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfs', ibT', ibUT', ls') \\
& | \\
& \text{spec-resolve:} \\
& cfs \neq [] \implies \\
& \text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs) \vee \text{is-Output } (\text{prog!}(pcOf\ (\text{last } cfs))) \vee \\
& \text{is-getInput } (\text{prog!}(pcOf\ (\text{last } cfs))) \implies \\
& pstate' = \text{update } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs) \implies \\
& cfs' = cfs \implies cfs' = \text{butlast } cfs \implies \\
& ibT = ibT' \implies ibUT = ibUT' \implies ls' = ls \\
& \implies \\
& (pstate, cfg, cfs, ibT, ibUT, ls) \rightarrow_S (pstate', cfs', cfs', ibT', ibUT', ls')
\end{aligned}$$

lemmas $\text{stepS-induct} = \text{stepS.induct}[\text{split-format}(\text{complete})]$

4.3.1 State Transitions

lemma $\text{stepS-nonspec-normal-iff}[\text{simp}]$:

$$\begin{aligned}
& cfs = [] \implies \neg \text{is-IfJump } (\text{prog!}(pcOf\ cfg)) \vee \neg \text{mispred } pstate [pcOf\ cfg] \\
& \implies \\
& (pstate, cfg, cfs, ibT, ibUT, ls) \rightarrow_S (pstate', cfs', cfs', ibT', ibUT', ls') \\
& \iff \\
& (pstate' = pstate \wedge \neg \text{finalB } (cfg, ibT, ibUT) \wedge \\
& \quad (cfs', ibT', ibUT') = \text{nextB } (cfg, ibT, ibUT) \wedge \\
& \quad cfs' = [] \wedge ls' = ls \cup \text{readLocs } cfg) \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma $\text{stepS-nonspec-normal-iff1}[\text{simp}]$:

$$\begin{aligned}
& cfs = [] \implies \neg \text{is-IfJump } (\text{prog!}pc) \vee \neg \text{mispred } pstate [pc] \\
& \implies \\
& (pstate, (\text{Config } pc\ (\text{State } (Vstore\ vs)\ \text{avst } h\ p)), cfs, ibT, ibUT, ls) \rightarrow_S (pstate', \\
& (\text{Config } pc'\ (\text{State } (Vstore\ vs')\ \text{avst}'\ h'\ p')), cfs', ibT', ibUT', ls') \\
& \iff \\
& (pstate' = pstate \wedge \neg \text{finalB } ((\text{Config } pc\ (\text{State } (Vstore\ vs)\ \text{avst } h\ p)), ibT, ibUT) \\
& \wedge \\
& \quad ((\text{Config } pc'\ (\text{State } (Vstore\ vs')\ \text{avst}'\ h'\ p')), ibT', ibUT') = \text{nextB } ((\text{Config } pc \\
& (\text{State } (Vstore\ vs)\ \text{avst } h\ p)), ibT, ibUT) \wedge \\
& \quad cfs' = [] \wedge ls' = ls \cup \text{readLocs } (\text{Config } pc\ (\text{State } (Vstore\ vs)\ \text{avst } h\ p)))
\end{aligned}$$

$\langle proof \rangle$

lemma *stepS-nonspec-mispred-iff[simp]*:

$cfgs = [] \implies is\text{-IfJump} (prog!(pcOf\ cfg)) \implies mispred\ pstate\ [pcOf\ cfg]$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \longleftrightarrow
 $(\exists\ cfg1'\ ibT1'\ ibUT1'. pstate' = update\ pstate\ [pcOf\ cfg] \wedge$
 $\neg\ finalM\ (cfg, ibT, ibUT) \wedge (cfg', ibT', ibUT') = nextB\ (cfg, ibT, ibUT) \wedge$
 $(cfg1', ibT1', ibUT1') = nextM\ (cfg, ibT, ibUT) \wedge$
 $cfgs' = [cfg1'] \wedge ls' = ls \cup readLocs\ cfg)$
 $\langle proof \rangle$

lemma *stepS-spec-normal-iff[simp]*:

$cfgs \neq [] \implies$
 $\neg\ resolve\ pstate\ (pcOf\ cfg \# map\ pcOf\ cfgs) \implies$
 $\neg\ is\text{-getInput} (prog!(pcOf\ (last\ cfgs))) \implies$
 $\neg\ is\text{-Output} (prog!(pcOf\ (last\ cfgs))) \implies$
 $\neg\ is\text{-IfJump} (prog!(pcOf\ (last\ cfgs))) \vee \neg\ mispred\ pstate\ (pcOf\ cfg \# map\ pcOf\$
 $cfgs) \implies$
 $prog!(pcOf\ (last\ cfgs)) \neq Fence$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \longleftrightarrow
 $(\exists\ cfg1'. pstate' = pstate \wedge$
 $\neg\ finalB\ (last\ cfgs, ibT, ibUT) \wedge (cfg1', ibT', ibUT') = nextB\ (last\ cfgs, ibT,$
 $ibUT) \wedge$
 $cfg' = cfg \wedge cfgs' = butlast\ cfgs\ @\ [cfg1'] \wedge ls' = ls \cup readLocs\ (last\ cfgs))$
 $\langle proof \rangle$

lemma *stepS-spec-mispred-iff[simp]*:

$cfgs \neq [] \implies$
 $\neg\ resolve\ pstate\ (pcOf\ cfg \# map\ pcOf\ cfgs) \implies$
 $is\text{-IfJump} (prog!(pcOf\ (last\ cfgs))) \implies mispred\ pstate\ (pcOf\ cfg \# map\ pcOf\ cfgs)$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \longleftrightarrow
 $(\exists\ cfg1'\ ibT1'\ ibUT1'\ lcfg'. pstate' = update\ pstate\ (pcOf\ cfg \# map\ pcOf\ cfgs) \wedge$
 $\neg\ finalM\ (last\ cfgs, ibT, ibUT) \wedge$
 $(lcfg', ibT', ibUT') = nextB\ (last\ cfgs, ibT, ibUT) \wedge$
 $(cfg1', ibT1', ibUT1') = nextM\ (last\ cfgs, ibT, ibUT) \wedge$
 $cfg' = cfg \wedge cfgs' = butlast\ cfgs\ @\ [lcfg']\ @\ [cfg1'] \wedge ls' = ls \cup readLocs\ (last$
 $cfgs))$
 $\langle proof \rangle$

lemma *stepS-spec-Fence-iff[simp]*:

$cfgs \neq [] \implies$

$\neg \text{resolve } pstate \text{ (pcOf } cfg \# \text{ map pcOf } cfgs) \implies$
 $\text{prog!}(\text{pcOf } (\text{last } cfgs)) = \text{Fence}$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \longleftrightarrow
 $(pstate' = pstate \wedge cfg = cfg' \wedge cfgs' = [] \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' =$
 $ls)$
 $\langle \text{proof} \rangle$

lemma *stepS-spec-resolve-iff[simp]*:

$cfgs \neq [] \implies$
 $\text{resolve } pstate \text{ (pcOf } cfg \# \text{ map pcOf } cfgs)$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \longleftrightarrow
 $(pstate' = \text{update } pstate \text{ (pcOf } cfg \# \text{ map pcOf } cfgs) \wedge$
 $cfg' = cfg \wedge cfgs' = \text{butlast } cfgs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' = ls)$
 $\langle \text{proof} \rangle$

lemma *stepS-cases[cases pred: stepS,*

consumes 1,

case-names nonspec-normal nonspec-mispred

spec-normal spec-mispred spec-Fence spec-resolve spec-resolveI spec-resolveO]:

assumes $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT',$
 $ls')$

obtains

$cfgs = []$
 $\neg \text{is-IfJump } (\text{prog!}(\text{pcOf } cfg)) \vee \neg \text{mispred } pstate \text{ [pcOf } cfg]$
 $pstate' = pstate$
 $\neg \text{finalB } (cfg, ibT, ibUT)$
 $(cfg', ibT', ibUT') = \text{nextB } (cfg, ibT, ibUT)$
 $cfgs' = []$
 $ls' = ls \cup \text{readLocs } cfg$

|

$cfgs = []$
 $\text{is-IfJump } (\text{prog!}(\text{pcOf } cfg)) \text{ mispred } pstate \text{ [pcOf } cfg]$
 $pstate' = \text{update } pstate \text{ [pcOf } cfg]$
 $\neg \text{finalM } (cfg, ibT, ibUT)$
 $(cfg', ibT', ibUT') = \text{nextB } (cfg, ibT, ibUT)$
 $\exists \text{cfg1}' \text{ ibT1}' \text{ ibUT1}'. (cfg1}', \text{ibT1}', \text{ibUT1}') = \text{nextM } (cfg, \text{ibT}, \text{ibUT})$
 $\wedge \text{cfgs}' = [\text{cfg1}']$
 $ls' = ls \cup \text{readLocs } cfg$

|

$cfgs \neq []$

$\neg \text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $\neg \text{is-IfJump } (prog!(pcOf (last\ cfs))) \vee \neg \text{mispred } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $prog!(pcOf (last\ cfs)) \neq Fence$
 $pstate' = pstate$
 $\neg \text{is-getInput } (prog!(pcOf (last\ cfs)))$
 $\neg \text{is-Output } (prog!(pcOf (last\ cfs)))$
 $cfg' = cfg$
 $ls' = ls \cup \text{readLocs } (last\ cfs)$
 $\exists cfg1'. \text{nextB } (last\ cfs, ibT, ibUT) = (cfg1', ibT', ibUT')$
 $\wedge cfs' = \text{butlast } cfs @ [cfg1']$

|

$cfs \neq []$
 $\neg \text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $\text{is-IfJump } (prog!(pcOf (last\ cfs))) \text{ mispred } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $pstate' = \text{update } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $\neg \text{finalM } (last\ cfs, ibT, ibUT)$
 $cfg' = cfg$
 $\exists lcfg' \text{ } cfg1' \text{ } ibT1' \text{ } ibUT1'.$
 $\text{nextB } (last\ cfs, ibT, ibUT) = (lcfg', ibT', ibUT') \wedge$
 $(cfg1', ibT1', ibUT1') = \text{nextM } (last\ cfs, ibT, ibUT) \wedge$
 $cfs' = \text{butlast } cfs @ [lcfg'] @ [cfg1']$
 $ls' = ls \cup \text{readLocs } (last\ cfs)$

|

$cfs \neq []$
 $\neg \text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $prog!(pcOf (last\ cfs)) = Fence$
 $pstate' = pstate$
 $cfg' = cfg$
 $cfs' = []$
 $ibT' = ibT$
 $ibUT' = ibUT$
 $ls' = ls$

|

$cfs \neq []$
 $\text{resolve } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $pstate' = \text{update } pstate (pcOf\ cfg \# \text{map } pcOf\ cfs)$
 $cfg' = cfg$
 $cfs' = \text{butlast } cfs$
 $ls' = ls$
 $ibT' = ibT$
 $ibUT' = ibUT$

|

$cfs \neq []$
 $\text{is-getInput } (prog!(pcOf (last\ cfs)))$

$pstate' = update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$
 $cfg' = cfg$
 $cfs' = butlast\ cfs$
 $ls' = ls$
 $ibT' = ibT$
 $ibUT' = ibUT$

|

$cfs \neq []$
 $is-Output\ (prog!(pcOf\ (last\ cfs)))$
 $pstate' = update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$
 $cfg' = cfg$
 $cfs' = butlast\ cfs$
 $ls' = ls$
 $ibT' = ibT$
 $ibUT' = ibUT$
 $\langle proof \rangle$

lemma $stepS-endPC$: $pcOf\ cfg = endPC \implies \neg (pstate, cfg, [], ibT, ibUT, ls) \rightarrow_S ss'$
 $\langle proof \rangle$

abbreviation

$stepsS :: configS \Rightarrow configS \Rightarrow bool$ (**infix** $\rightarrow_S^* 55$)
where $x \rightarrow_S^* y \equiv star\ stepS\ x\ y$

definition $finalS = final\ stepS$

lemmas $finalS-defs = final-def\ finalS-def$

lemma $stepS-determ$:

$cfg-ib \rightarrow_S cfg-ib' \implies cfg-ib \rightarrow_S cfg-ib'' \implies cfg-ib'' = cfg-ib'$
 $\langle proof \rangle$

lemma $stepS-0$: $(pstate, Config\ 0\ s, [], ibT, ibUT, ls) \rightarrow_S (pstate, Config\ 1\ s, [], ibT, ibUT, ls)$
 $\langle proof \rangle$

lemma $stepS-imp-stepB$: $(pstate, cfg, [], ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfs', ibT', ibUT', ls') \implies (cfg, ibT, ibUT) \rightarrow_B (cfg', ibT', ibUT')$
 $\langle proof \rangle$

4.3.2 Elimination Rules

lemma $stepS-Assign2E$:

assumes $\langle ps3, cfg3, cfs3, ibT3, ibUT3, ls3 \rangle \rightarrow_S (ps3', cfg3', cfs3', ibT3', ibUT3', ls3')$
and $\langle ps4, cfg4, cfs4, ibT4, ibUT4, ls4 \rangle \rightarrow_S (ps4', cfg4', cfs4', ibT4', ibUT4', ls4')$
and $\langle cfg3 = (Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)) \rangle$ **and** $\langle cfg3' =$

```

(Config pc3' (State (Vstore vs3') avst3' h3' p3'))
  and ⟨cfg4 = (Config pc4 (State (Vstore vs4) avst4 h4 p4))⟩ and ⟨cfg4' =
(Config pc4' (State (Vstore vs4') avst4' h4' p4'))⟩
  and ⟨cfs3 = []⟩ and ⟨cfs4 = []⟩
  and ⟨prog!pc3 = (x ::= a)⟩ and ⟨pcOf cfs3 = pcOf cfs4⟩
shows ⟨cfs3' = [] ∧ cfs4' = [] ∧
vs3' = (vs3(x := aval a (stateOf cfs3))) ∧
vs4' = (vs4(x := aval a (stateOf cfs4))) ∧
pc3' = Suc pc3 ∧ pc4' = Suc pc4 ∧ ls4' = ls4 ∪ readLocs cfs4 ∧
avst3' = avst3 ∧ avst4' = avst4 ∧ ls3' = ls3 ∪ readLocs cfs3 ∧
p3 = p3' ∧ p4 = p4'⟩
⟨proof⟩

```

end

end

5 Relative Security instantiation - Common Aspects

This theory sets up a generic instantiation infrastructure for all our running examples. For a detailed explanation of each example and it's (dis)proof of Relative Security see the work by Dongol et al. [2]

```

theory Instance-Common
imports ../IMP/Step-Normal ../IMP/Step-Spec
begin

```

```

no-notation bot ( $\perp$ )

```

```

abbreviation noninform ( $\perp$ ) where  $\perp \equiv \text{undefined}$ 

```

```

declare split-paired-All[simp del]
declare split-paired-Ex[simp del]

```

```

definition noMisSpec where noMisSpec (cfs::config list)  $\equiv$  (cfs = [])
lemma noMisSpec-ext[simp]:map x cfs = map x cfs'  $\implies$  noMisSpec cfs  $\longleftrightarrow$ 
noMisSpec cfs'
  ⟨proof⟩

```

```

definition misSpecL1 where misSpecL1 (cfs::config list)  $\equiv$  (length cfs = Suc

```

0)

lemma *misSpecL1-len:misSpecL1* $cfgs \longleftrightarrow length\ cfgs = 1$ *<proof>*

lemma *misSpecL1-ne:misSpecL1* $cfgs \implies cfgs \neq []$ *<proof>*

definition *misSpecL2* **where** *misSpecL2* ($cfgs::config\ list$) $\equiv (length\ cfgs = 2)$

fun *tuple*:: $'a \times 'b \times 'c \Rightarrow 'a \times 'b$
where *tuple* (a,b,c) = (a,b)

fun *tuple-sel*:: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'b \times 'd$
where *tuple-sel* (a,b,c,d,e) = (b,d)

fun *cfgsOf*:: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'c$
where *cfgsOf* (a,b,c,d,e) = c

fun *pstateOf*:: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$
where *pstateOf* (a,b,c,d,e) = a

fun *stateOfs*:: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'b$
where *stateOfs* (a,b,c,d,e) = b

context *Prog-Mispred*

begin

The "vanilla-semantics" transitions are the normal executions (featuring no speculation):

Vanilla-semantics system model: given by the normal semantics

type-synonym *stateV* = $config \times val\ llist \times val\ llist \times loc\ set$

fun *validTransV* **where** *validTransV* ($cfg\text{-}ib\text{-}ls, cfg\text{-}ib\text{-}ls'$) = $cfg\text{-}ib\text{-}ls \rightarrow N\ cfg\text{-}ib\text{-}ls'$

Vanilla-semantics observation infrastructure (part of the vanilla-semantics state-wise attacker model):

The attacker observes the output value, the program counter history and the set of accessed locations so far:

type-synonym *obsV* = $val \times loc\ set$

The attacker-action is just a value (used as input to the function):

type-synonym *actV* = val

The attacker's interaction

fun *isIntV* :: *stateV* ⇒ *bool* **where**
isIntV ss = (¬ *finalN ss*)

The attacker interacts with the system by passing input to the function and reading the outputs (standard channel) and the accessed locations (side channel)

fun *getIntV* :: *stateV* ⇒ *actV* × *obsV* **where**
getIntV (cfg, ibT, ibUT, ls) =
 (*case prog!(pcOf cfg)* of
 | *Input T* - ⇒ (*lhd ibT*, ⊥)
 | *Input U* - ⇒ (*lhd ibUT*, ⊥)
 | *Output U* - ⇒ (⊥, (*outOf (prog!(pcOf cfg)) (stateOf cfg), ls*))
 | - ⇒ (⊥, ⊥)
)

lemma *validTransV-iff-nextN*: *validTransV (s1, s2)* = (¬ *finalN s1* ∧ *nextN s1* = *s2*)
 ⟨*proof*⟩

The optimization-enhanced semantics system model: given by the speculative semantics

type-synonym *stateO* = *configS*
fun *validTransO* **where** *validTransO (cfgS, cfgS')* = *cfgS* →*S* *cfgS'*

Optimization-enhanced semantics observation infrastructure (part of the optimization-enhanced semantics state-wise attacker model): similar to that of the vanilla semantics, in that the standard-channel inputs and outputs are those produced by the normal execution. However, the side-channel outputs (the sets of read locations) are also collected.

type-synonym *obsO* = *val* × *loc set*
type-synonym *actO* = *val*
fun *isIntO* :: *stateO* ⇒ *bool* **where**
isIntO ss = (¬ *finalS ss*)
fun *getIntO* :: *stateO* ⇒ *actO* × *obsO* **where**
getIntO (pstate, cfg, cfgs, ibT, ibUT, ls) =
 (*case (cfgs, prog!(pcOf cfg))* of
 | ([], *Input T* -) ⇒ (*lhd ibT*, ⊥)
 | ([], *Input U* -) ⇒ (*lhd ibUT*, ⊥)
 | ([], *Output U* -) ⇒
 (⊥, (*outOf (prog!(pcOf cfg)) (stateOf cfg), ls*))
 | - ⇒ (⊥, ⊥)
)

end

locale *Prog-Mispred-Init* =
Prog-Mispred prog mispred resolve update

```

for prog :: com list
and mispred :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
and resolve :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
and update :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  predState
+
fixes initPstate :: predState
  and istate :: state  $\Rightarrow$  bool
begin

```

```

fun istateV :: stateV  $\Rightarrow$  bool where
istateV (cfg, ibT, ibUT, ls)  $\longleftrightarrow$ 
  pcOf cfg = 0  $\wedge$  istate (stateOf cfg)  $\wedge$ 
  llength ibT =  $\infty$   $\wedge$  llength ibUT =  $\infty$   $\wedge$ 
  ls = {}

```

```

fun istateO :: stateO  $\Rightarrow$  bool where
istateO (pstate, cfg, cfs, ibT, ibUT, ls)  $\longleftrightarrow$ 
  pstate = initPstate  $\wedge$ 
  pcOf cfg = 0  $\wedge$  ls = {}  $\wedge$ 
  istate (stateOf cfg)  $\wedge$ 
  cfs = []  $\wedge$  llength ibT =  $\infty$   $\wedge$  llength ibUT =  $\infty$ 

```

lemma *istateV-config-imp*:
 $istateV (cfg, ibT, ibUT, ls) \implies pcOf\ cfg = 0 \wedge ls = \{\} \wedge ibT \neq LNil$
<proof>

lemma *istateO-config-imp*:
 $istateO (pstate, cfg, cfs, ibT, ibUT, ls) \implies$
 $cfs = [] \wedge pcOf\ cfg = 0 \wedge ls = \{\} \wedge ibT \neq LNil$
<proof>

definition *same-var-all* $x\ cfg1\ cfg2\ cfg3\ cfs3\ cfg4\ cfs4 \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg1))\ x = vstore\ (getVstore\ (stateOf\ cfg4))\ x \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg2))\ x = vstore\ (getVstore\ (stateOf\ cfg4))\ x \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg3))\ x = vstore\ (getVstore\ (stateOf\ cfg4))\ x \wedge$
 $(\forall\ cfg3' \in set\ cfs3.\ vstore\ (getVstore\ (stateOf\ cfg3'))\ x = vstore\ (getVstore\ (stateOf\$
 $cfg3))\ x) \wedge$
 $(\forall\ cfg4' \in set\ cfs4.\ vstore\ (getVstore\ (stateOf\ cfg4'))\ x = vstore\ (getVstore\ (stateOf\$
 $cfg4))\ x)$

definition *same-var* $x\ cfg\ cfg' \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg))\ x = vstore\ (getVstore\ (stateOf\ cfg'))\ x$

definition *same-var-val* x ($val::int$) $cfg\ cfg' \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg))\ x = vstore\ (getVstore\ (stateOf\ cfg'))\ x \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg))\ x = val$

definition *same-var-o* $ii\ cfg3\ cfs3\ cfg4\ cfs4 \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg3))\ ii = vstore\ (getVstore\ (stateOf\ cfg4))\ ii \wedge$
 $(\forall\ cfg3' \in set\ cfs3. vstore\ (getVstore\ (stateOf\ cfg3'))\ ii = vstore\ (getVstore\ (stateOf\ cfg3))\ ii) \wedge$
 $(\forall\ cfg4' \in set\ cfs4. vstore\ (getVstore\ (stateOf\ cfg4'))\ ii = vstore\ (getVstore\ (stateOf\ cfg4))\ ii)$

lemma *set-var-shrink*: $\forall\ cfg3' \in set\ cfs.$
 $vstore\ (getVstore\ (stateOf\ cfg3'))\ var =$
 $vstore\ (getVstore\ (stateOf\ cfg))\ var$
 \implies
 $\forall\ cfg3' \in set\ (butlast\ cfs).$
 $vstore\ (getVstore\ (stateOf\ cfg3'))\ var =$
 $vstore\ (getVstore\ (stateOf\ cfg))\ var$
 $\langle proof \rangle$

lemma *heapSimp*: $(\forall\ cfg'' \in set\ cfs''. getHheap\ (stateOf\ cfg') = getHheap\ (stateOf\ cfg'')) \wedge cfs'' \neq []$
 $\implies getHheap\ (stateOf\ cfg') = getHheap\ (stateOf\ (last\ cfs''))$
 $\langle proof \rangle$

lemma *heapSimp2*: $(\forall\ cfg'' \in set\ cfs''. getHheap\ (stateOf\ cfg') = getHheap\ (stateOf\ cfg'')) \wedge cfs'' \neq []$
 $\implies getHheap\ (stateOf\ cfg') = getHheap\ (stateOf\ (hd\ cfs''))$
 $\langle proof \rangle$

lemma *array-baseSimp*: $array-base\ aa1\ (getAvstore\ (stateOf\ cfg)) =$
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg')) \wedge$
 $(\forall\ cfg' \in set\ cfs. array-base\ aa1\ (getAvstore\ (stateOf\ cfg')) =$
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg)))$
 $\wedge\ cfs \neq []$
 \implies
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg)) =$
 $array-base\ aa1\ (getAvstore\ (stateOf\ (last\ cfs)))$
 $\langle proof \rangle$

lemma *finalB-imp-finalS*: $finalB\ (cfg, ibT, ibUT) \implies (\forall\ pstate\ cfs\ ls. finalS\ (pstate, cfg, [], ibT, ibUT, ls))$

<proof>

lemma *cfgs-Suc-zero*[simp]: $length\ cfgs = Suc\ 0 \implies cfgs = [last\ cfgs]$
<proof>

lemma *cfgs-map*[simp]: $length\ cfgs = Suc\ 0 \implies map\ pcOf\ cfgs = [pcOf\ (last\ cfgs)]$
<proof>

lemma *is-misSpecL1*[simp]: $misSpecL1\ [a]$ *<proof>*

end

end

6 Relative Security Instance: Secret Memory

This theory sets up an instance of Relative Security with the secrets as the initial memories

theory *Instance-Secret-IMem*
imports *Instance-Common Relative-Security.Relative-Security*
begin

no-notation *bot* (\perp)
type-synonym *secret* = *state*

context *Prog-Mispred*
begin

fun *corrState* :: $stateV \Rightarrow stateO \Rightarrow bool$ **where**
corrState *cfgO* *cfgA* = *True*

Since all our programs will have "Start" followed by the rest, with the rest not containing "Start". The secret will be "uploaded" at this Start moment.

definition *isSecV* :: $stateV \Rightarrow bool$ **where**
isSecV *ss* $\equiv case\ ss\ of\ (cfg,ibT,ibUT) \Rightarrow (pcOf\ cfg = 0)$

We consider the entire initial state as a secret:

fun *getSecV* :: $stateV \Rightarrow secret$ **where**
getSecV (*cfg,ibT,ibUT*) = *stateOf* *cfg*

The secrecy infrastructure is similar to that of the "original" semantics:

definition *isSecO* :: $stateO \Rightarrow bool$ **where**
isSecO *ss* $\equiv case\ ss\ of\ (pstate,cfg,cfgs,ibT,ibUT,ls) \Rightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
fun *getSecO* :: $stateO \Rightarrow secret$ **where**
getSecO (*pstate,cfg,cfgs,ibT,ibUT,ls*) = *stateOf* *cfg*

lemma *isSecV-iff:isSecV ss* \longleftrightarrow *pcOf (fst ss) = 0*
 ⟨proof⟩

lemma *validTransO-iff-nextS*: *validTransO (s1, s2) =* $(\neg \text{finalS } s1 \wedge (\text{stepS } s1 \text{ } s2))$
 ⟨proof⟩

end

sublocale *Prog-Mispred-Init < Rel-Sec* **where**
validTransV = validTransV **and** *istateV = istateV*
and *finalV = finalN*
and *isSecV = isSecV* **and** *getSecV = getSecV*
and *isIntV = isIntV* **and** *getIntV = getIntV*

and *validTransO = validTransO* **and** *istateO = istateO*
and *finalO = finalS*
and *isSecO = isSecO* **and** *getSecO = getSecO*
and *isIntO = isIntO* **and** *getIntO = getIntO*
and *corrState = corrState*
 ⟨proof⟩

context *Prog-Mispred-Init*
begin

lemmas *reachV-induct = Van.reach.induct[split-format(complete)]*
lemmas *reachO-induct = Opt.reach.induct[split-format(complete)]*

lemma *is-getTrustedInput-getActV[simp]*:
(prog!(pcOf cfg)) = Input T s \implies *getActV (cfg,ibT,ibUT,ls) = lhd ibT*
 ⟨proof⟩

lemma *not-is-getTrustedInput-getActV[simp]*:
 \neg *is-getInput (prog!(pcOf cfg))* \implies *getActV (cfg,ibT,ibUT,ls) = noninform*
 ⟨proof⟩

lemma *is-Output-getObsV[simp]*:
(prog!(pcOf cfg)) = Output U out \implies *getObsV (cfg,ibT,ibUT,ls) =*
(outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
 ⟨proof⟩

lemma *not-is-Output-getObsV[simp]*:
 \neg *is-Output (prog!(pcOf cfg))* \implies *getObsV (cfg,ibT,ibUT,ls) = \perp*
 ⟨proof⟩

lemma *is-getTrustedInput-Nil-getActO[simp]*:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ } s \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd ibT}$
 $\langle \text{proof} \rangle$

lemma *not-is-getTrustedInput-Nil-getActO[simp]*:

$\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg}))$
 $\vee \text{cfgs} \neq [] \implies \text{getActO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
 $\langle \text{proof} \rangle$

lemma *is-Output-Nil-getObsO[simp]*:

$\text{prog!}(\text{pcOf } \text{cfg}) = \text{Output } U \text{ } s \implies$
 $\text{getObsO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = (\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
 $\langle \text{proof} \rangle$

lemma *not-is-Output-Nil-getObsO[simp]*:

$\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \vee \text{cfgs} \neq [] \implies \text{getObsO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls})$
 $= \perp$
 $\langle \text{proof} \rangle$

lemma *getActV-simps*:

$\text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{case } \text{prog!}(\text{pcOf } \text{cfg}) \text{ of}$
 $\quad \text{Input } T \text{ } - \Rightarrow \text{lhd ibT}$
 $\quad | \text{Input } U \text{ } - \Rightarrow \text{lhd ibUT}$
 $\quad | - \Rightarrow \perp$
 $)$
 $\langle \text{proof} \rangle$

lemma *getObsV-simps*:

$\text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{case } \text{prog!}(\text{pcOf } \text{cfg}) \text{ of}$
 $\quad \text{Output } U \text{ } - \Rightarrow (\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
 $\quad | - \Rightarrow \perp$
 $)$
 $\langle \text{proof} \rangle$

lemma *getActO-simps*:

$\text{getActO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{case } (\text{cfgs}, \text{prog!}(\text{pcOf } \text{cfg})) \text{ of}$
 $\quad ([], \text{Input } T \text{ } -) \Rightarrow \text{lhd ibT}$
 $\quad | ([], \text{Input } U \text{ } -) \Rightarrow \text{lhd ibUT}$
 $\quad | - \Rightarrow \perp$
 $)$
 $\langle \text{proof} \rangle$

lemma *getObsO-simps*:

```

getObsO (pstate, cfg, cfs, ibT, ibUT, ls) =
  (case (cfs, prog!(pcOf cfg)) of
    ([], Output U -) => (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
    | - => ⊥
  )
  ⟨proof⟩

```

end

end

7 Relative Security Instance: Secret Memory Input

This theory sets up an instance of Relative Security used to prove an Security of a potentially infinite program

```

theory Instance-Secret-IMem-Inp
  imports Instance-Common Relative-Security.Relative-Security
begin

```

Using the following notation to denote an undefined element

no-notation *bot* ($\langle \perp \rangle$)

definition *ffile* :: *vname* **where** *ffile* = "ffile"

definition *xx* :: *vname* **where** *xx* = "x"

definition *yy* :: *vname* **where** *yy* = "yy"

type-synonym *secret* = *state* × *val* × *val*

abbreviation *writeSecretOnFile* **where** *writeSecretOnFile* ≡ (*Output* *T* (*Fun* (*V* *xx*) (*V* *yy*)))

lemma *writeOnFile-not-Jump[simp]*: \neg *is-IfJump* *writeSecretOnFile* ⟨proof⟩

lemma *writeOnFile-not-Inp[simp]*: \neg *is-getInput* *writeSecretOnFile* ⟨proof⟩

lemma *writeOnFile-not-Fence[simp]*: *writeSecretOnFile* ≠ *Fence* ⟨proof⟩

definition *ffileVal* **where** *ffileVal* *cfg* = *vstoreOf*(*cfg*) *ffile*

lemma *ffileVal-vstore[simp]*: *ffileVal* *cfg* = *vstoreOf*(*cfg*) *ffile* ⟨proof⟩

context *Prog-Mispred*

begin

The following functions and definitions make up the required components of the Relative Security locale

```

fun corrState :: stateV => stateO => bool where
corrState cfgO cfgA = True

```

definition $isSecV :: stateV \Rightarrow bool$ **where**
 $isSecV\ ss \equiv case\ ss\ of\ (cfg,ibT,ibUT,ls) \Rightarrow \neg finalN\ ss$

fun $getSecV :: stateV \Rightarrow secret$ **where**
 $getSecV\ (cfg,ibT,ibUT,ls) =$
 (case $prog!(pcOf\ cfg)$ of
 $Start \Rightarrow (stateOf\ cfg, \perp, \perp)$
 $| Input\ T - \Rightarrow (\perp, lhd\ ibT, \perp)$
 $| - \Rightarrow (\perp, \perp, \perp)$)

lemma $isSecV\text{-iff}: isSecV\ ss \longleftrightarrow \neg finalN\ ss$
 {proof}

definition $isSecO :: stateO \Rightarrow bool$ **where**
 $isSecO\ ss \equiv case\ ss\ of\ (pstate, cfg, cfgs, ibT, ibUT, ls) \Rightarrow \neg finalS\ ss \wedge cfgs = []$
fun $getSecO :: stateO \Rightarrow secret$ **where**
 $getSecO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 (case $prog!(pcOf\ cfg)$ of
 $Start \Rightarrow (stateOf\ cfg, \perp, \perp)$
 $| Input\ T - \Rightarrow (\perp, lhd\ ibT, \perp)$
 $| - \Rightarrow (\perp, \perp, \perp)$)
end

sublocale $Prog\text{-Mispred}\text{-Init} < Rel\text{-Sec}$ **where**
 $validTransV = validTransV$ **and** $istateV = istateV$
and $finalV = finalN$
and $isSecV = isSecV$ **and** $getSecV = getSecV$
and $isIntV = isIntV$ **and** $getIntV = getIntV$

and $validTransO = validTransO$ **and** $istateO = istateO$
and $finalO = finalS$
and $isSecO = isSecO$ **and** $getSecO = getSecO$
and $isIntO = isIntO$ **and** $getIntO = getIntO$
and $corrState = corrState$
 {proof}

context $Prog\text{-Mispred}\text{-Init}$
begin

lemmas *reachV-induct* = *Van.reach.induct*[*split-format*(*complete*)]

lemmas *reachO-induct* = *Opt.reach.induct*[*split-format*(*complete*)]

lemma *is-getInputT-getActV*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } U \text{ inp} \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibUT}$
<proof>

lemma *is-getInputU-getActV*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ inp} \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibT}$
<proof>

lemma *not-is-getInput-getActV*[*simp*]:

$\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
<proof>

lemma *is-Output-getObsV*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Output } U \text{ out} \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
<proof>

lemma *not-is-Output-getObsV*[*simp*]:

$\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
<proof>

lemma *is-getInputT-Nil-getActO*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ inp} \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibT}$
<proof>

lemma *is-getInputU-Nil-getActO*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } U \text{ inp} \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibUT}$
<proof>

lemma *not-is-getInput-Nil-getActO*[*simp*]:

$(\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg})))$
 $\vee \text{cfgs} \neq [] \implies \text{getActO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
<proof>

lemma *is-Output-Nil-getObsO*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Output } U \text{ out} \implies$
 $\text{getObsO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = (\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
<proof>

lemma *not-is-Output-Nil-getObsO*[*simp*]:

$\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \vee \text{cfgs} \neq [] \implies \text{getObsO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls})$
 $= \perp$
<proof>

lemma *getActV-simps*:
 $getActV\ (cfg,ibT,ibUT,ls) =$
 (*case prog!(pcOf cfg) of*
 Input T - \Rightarrow lhd ibT
 | *Input U - \Rightarrow lhd ibUT*
 | - $\Rightarrow \perp$
)
<proof>

lemma *getObsV-simps*:
 $getObsV\ (cfg,ibT,ibUT,ls) =$
 (*case prog!(pcOf cfg) of*
 Output U - \Rightarrow (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
 | - $\Rightarrow \perp$
)
<proof>

lemma *getActO-simps*:
 $getActO\ (pstate,cfg,cfgs,ibT,ibUT,ls) =$
 (*case (cfgs,prog!(pcOf cfg)) of*
 (\square , *Input T - \Rightarrow lhd ibT*)
 | (\square , *Input U - \Rightarrow lhd ibUT*)
 | - $\Rightarrow \perp$
)
<proof>

lemma *getObsO-simps*:
 $getObsO\ (pstate,cfg,cfgs,ibT,ibUT,ls) =$
 (*case (cfgs,prog!(pcOf cfg)) of*
 (\square , *Output U - \Rightarrow (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)*)
 | - $\Rightarrow \perp$
)
<proof>

end

end

8 Disproof of Relative Security for fun1

theory *Fun1*
imports *../Instance-IMP/Instance-Secret-IMem*
Secret-Directed-Unwinding.SD-Unwinding-fn

begin

8.1 Function definition and Boilerplate

no-notation *bot* ($\langle \perp \rangle$)

consts *NN* :: *nat*

consts *input* :: *int*

definition *aa1* :: *avname* **where** *aa1* = "a1"

definition *aa2* :: *avname* **where** *aa2* = "a2"

definition *vv* :: *avname* **where** *vv* = "v"

definition *xx* :: *avname* **where** *xx* = "i"

definition *tt* :: *avname* **where** *tt* = "t"

lemma *NN-suc*[*simp*]:*nat* (*NN* + 1) = *Suc* (*nat* *NN*)
<proof>

lemma *NN:NN* ≥ 0 *<proof>*

lemmas *vvars-defs* = *aa1-def aa2-def vv-def xx-def tt-def*

lemma *vvars-dff*[*simp*]:

aa1 ≠ *aa2* *aa1* ≠ *vv* *aa1* ≠ *xx* *aa1* ≠ *tt*

aa2 ≠ *aa1* *aa2* ≠ *vv* *aa2* ≠ *xx* *aa2* ≠ *tt*

vv ≠ *aa1* *vv* ≠ *aa2* *vv* ≠ *xx* *vv* ≠ *tt*

xx ≠ *aa1* *xx* ≠ *aa2* *xx* ≠ *vv* *xx* ≠ *tt*

tt ≠ *aa1* *tt* ≠ *aa2* *tt* ≠ *vv* *tt* ≠ *xx*

<proof>

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

definition *s-add* = {*a*. *a* ≠ *nat* *NN*+1}

fun *vs₀*::*char list* ⇒ *int* **where**

vs₀ *x* = 0

lemma *vs0*[*simp*]:(λx . 0) = *vs₀* *<proof>*

fun *as*:: *char list* ⇒ *nat* × *nat* **where**

as *a* = (if *a* = *aa1* then (0, *nat* *NN*)

else (if *a* = *aa2* then (*nat* *NN*, *nat* *size-aa2*)

else (*nat* *size-aa2*, 0)))

definition *avst'* ≡ (*Avstore as*)

lemmas *avst-defs* = *avst'-def as.simps*

lemma *avstore-loc[simp]*: *Avstore* ($\lambda a.$ if $a = aa1$ then $(0, \text{nat } NN)$ else if $a = aa2$ then $(\text{nat } NN, \text{nat size-aa2})$ else $(\text{nat size-aa2}, 0)$) =
 $avst'$
 $\langle proof \rangle$

abbreviation *read-add* $\equiv \{a. a \neq (\text{nat } NN + 1)\}$

fun *initVstore* :: *vstore* \Rightarrow *bool* **where**
initVstore (*Vstore* *vst*) = (*vst* = *vs₀*)

fun *initAvstore* :: *avstore* \Rightarrow *bool* **where**
initAvstore *avst* = (*avst* = *avst'*)
fun *initHeap*::(*nat* \Rightarrow *int*) \Rightarrow *bool* **where**
initHeap *h* = ($\forall x \in \text{read-add}. h\ x = 0$)

lemma *initAvstore-0[intro]*: *initAvstore* *avst'* \Longrightarrow *array-base* *aa1* *avst'* = 0
 $\langle proof \rangle$

fun *istate* :: *state* \Rightarrow *bool* **where**
istate *s* =
(*initVstore* (*getVstore* *s*) \wedge
initAvstore (*getAvstore* *s*) \wedge
initHeap (*getHheap* *s*))

definition *prog* \equiv
[
 \emptyset *Start* ,
 $\not\! /$ *Input* *U* *xx* ,
 $\not\! /$ *tt* ::= (*N* 0),
 $\not\! /$ *IfJump* (*Less* (*V* *xx*) (*N* *NN*)) 4 5,
 $\not\! /$ *tt* ::= (*VA* *aa2* (*Times* (*VA* *aa1* (*V* *xx*)) (*N* 512))),
 $\not\! /$ *Output* *U* (*V* *tt*)
]

lemma *cases-5*: (*i*::*pcounter*) = 0 \vee *i* = 1 \vee *i* = 2 \vee *i* = 3 \vee *i* = 4 \vee *i* = 5 \vee *i* > 5
 $\langle proof \rangle$

lemma *xx-NN-cases*: *vs* *xx* < (*int* *NN*) \vee *vs* *xx* \geq (*int* *NN*) $\langle proof \rangle$

lemma *is-If-pcOf[simp]*:
pcOf *cfg* < 6 \Longrightarrow *is-IfJump* (*prog* ! (*pcOf* *cfg*)) \longleftrightarrow *pcOf* *cfg* = 3
 $\langle proof \rangle$

lemma *is-If-pc[simp]*:

$pc < 6 \implies is\text{-IfJump} (prog ! pc) \longleftrightarrow pc = 3$
(proof)

lemma *eq-Fence-pc[simp]*:
 $pc < 6 \implies prog ! pc \neq Fence$
(proof)

fun *mispred* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool* **where**
mispred *p pc* = (if *pc* = [3] then *True* else *False*)

fun *resolve* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool* **where**
resolve *p pc* = (if *pc* = [5,5] then *True* else *False*)

consts *update* :: *predState* \Rightarrow *pcounter list* \Rightarrow *predState*
consts *pstate₀* :: *predState*

interpretation *Prog-Mispred-Init* **where**
prog = *prog* **and** *initPstate* = *pstate₀* **and**
mispred = *mispred* **and** *resolve* = *resolve* **and** *update* = *update* **and**
istate = *istate*
(proof)

abbreviation
stepB-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow
bool (**infix** $\langle \rightarrow B \rangle$ 55)
where $x \rightarrow B y == stepB x y$

abbreviation
stepsB-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow
bool (**infix** $\langle \rightarrow B^* \rangle$ 55)
where $x \rightarrow B^* y == star stepB x y$

abbreviation
stepM-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow
bool (**infix** $\langle \rightarrow M \rangle$ 55)
where $x \rightarrow M y == stepM x y$

abbreviation
stepN-abbrev :: *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *bool* (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == stepN x y$

abbreviation

$stepsN\text{-abbrev} :: config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist \times loc\ set \Rightarrow bool$ (**infix** $\langle \rightarrow N^* \rangle$ 55)
where $x \rightarrow N^* y == star\ stepN\ x\ y$

abbreviation

$stepS\text{-abbrev} :: configS \Rightarrow configS \Rightarrow bool$ (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == stepS\ x\ y$

abbreviation

$stepsS\text{-abbrev} :: configS \Rightarrow configS \Rightarrow bool$ (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == star\ stepS\ x\ y$

lemma $endPC[simp]: endPC = 6$

$\langle proof \rangle$

lemma $is\text{-getTrustedInput}\text{-}pcOf[simp]: pcOf\ cfg < 6 \Longrightarrow is\text{-getInput}\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 1$

$\langle proof \rangle$

lemma $getTrustedInput\text{-}pcOf[simp]: (prog!1) = Input\ U\ xx$

$\langle proof \rangle$

lemma $is\text{-Output}\text{-}pcOf[simp]: pcOf\ cfg < 6 \Longrightarrow is\text{-Output}\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 5 \vee pcOf\ cfg = 6$

$\langle proof \rangle$

lemma $is\text{-Fence}\text{-}pcOf[simp]: pcOf\ cfg < 6 \Longrightarrow (prog!(pcOf\ cfg)) \neq Fence$

$\langle proof \rangle$

lemma $prog0[simp]: prog\ !\ 0 = Start$

$\langle proof \rangle$

lemma $prog1[simp]: prog\ !\ (Suc\ 0) = Input\ U\ xx$

$\langle proof \rangle$

lemma $prog2[simp]: prog\ !\ 2 = tt ::= (N\ 0)$

$\langle proof \rangle$

lemma $prog3[simp]: prog\ !\ 3 = IfJump\ (Less\ (V\ xx)\ (N\ NN))\ 4\ 5$

$\langle proof \rangle$

lemma $prog4[simp]: prog\ !\ 4 = tt ::= (VA\ aa2\ (Times\ (VA\ aa1\ (V\ xx))\ (N\ 512)))$

$\langle proof \rangle$

lemma *prog5[simp]:prog ! 5 = Output U (V tt)*
 $\langle proof \rangle$

lemma *isSecV-pcOf[simp]:*
 $isSecV (cfg,ibT, ibUT) \longleftrightarrow pcOf\ cfg = 0$
 $\langle proof \rangle$

lemma *isSecO-pcOf[simp]:*
 $isSecO (pstate,cfg,cfgs,ibT,ibUT,ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfgs = [])$
 $\langle proof \rangle$

lemma *getInputT-not[simp]: pcOf cfg < 6 \implies*
 $(prog ! pcOf\ cfg) \neq Input\ T\ x$
 $\langle proof \rangle$

lemma *getActV-pcOf[simp]:*
 $pcOf\ cfg < 6 \implies$
 $getActV (cfg,ibT,ibUT,ls) =$
 $(if\ pcOf\ cfg = 1\ then\ lhd\ ibUT\ else\ \perp)$
 $\langle proof \rangle$

lemma *getObsV-pcOf[simp]:*
 $pcOf\ cfg < 6 \implies$
 $getObsV (cfg,ibT,ibUT,ls) =$
 $(if\ pcOf\ cfg = 5\ then$
 $(outOf (prog!(pcOf\ cfg)) (stateOf\ cfg), ls)$
 $else\ \perp$
 $)$
 $\langle proof \rangle$

lemma *getActO-pcOf[simp]:*
 $pcOf\ cfg < 6 \implies$
 $getActO (pstate,cfg,cfgs,ibT,ibUT,ls) =$
 $(if\ pcOf\ cfg = 1 \wedge cfgs = []\ then\ lhd\ ibUT\ else\ \perp)$
 $\langle proof \rangle$

lemma *getObsO-pcOf[simp]:*
 $pcOf\ cfg < 6 \implies$
 $getObsO (pstate,cfg,cfgs,ibT,ibUT,ls) =$
 $(if\ (pcOf\ cfg = 5 \wedge cfgs = [])\ then$
 $(outOf (prog!(pcOf\ cfg)) (stateOf\ cfg), ls)$
 $else\ \perp$
 $)$
 $\langle proof \rangle$

lemma *nextB-pc0[simp]*:
 $nextB (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc0[simp]*:
 $readLocs (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc1[simp]*:
 $ibUT \neq LNil \implies nextB (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p), ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc1[simp]*:
 $readLocs (Config\ 1\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc1'[simp]*:
 $ibUT \neq LNil \implies nextB (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p), ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc1'[simp]*:
 $readLocs (Config\ (Suc\ 0)\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc2[simp]*:
 $nextB (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $((Config\ 3\ (State\ (Vstore\ (vs(tt := 0))))\ avst\ h\ p), ibT, ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc2[simp]*:
 $readLocs (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc3-then[simp]*:
 $vs\ xx < NN \implies$
 $nextB (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$

(*Config 4* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *nextB-pc3-else*[*simp*]:

vs xx ≥ NN ⇒
nextB (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
 (*Config 5* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *nextB-pc3*:

nextB (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
 (*Config* (*if vs xx < NN then 4 else 5*) (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *nextM-pc3-then*[*simp*]:

vs xx ≥ NN ⇒
nextM (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
 (*Config 4* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *nextM-pc3-else*[*simp*]:

vs xx < NN ⇒
nextM (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
 (*Config 5* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *nextM-pc3*:

nextM (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
 (*Config* (*if vs xx < NN then 5 else 4*) (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *readLocs-pc3*[*simp*]:

readLocs (*Config 3 s*) = {}
 ⟨*proof*⟩

lemma *nextB-pc4*[*simp*]:

nextB (*Config 4* (*State* (*Vstore vs*) *avst (Heap h) p*), *ibT*, *ibUT*) =
 (*let i = array-loc aa1 (nat (vs xx)) avst; j = (array-loc aa2 (nat ((h i) * 512))*
avst)
in (*Config 5* (*State* (*Vstore (vs(tt := h j))*) *avst (Heap h) p*), *ibT*, *ibUT*)
 ⟨*proof*⟩

lemma *readLocs-pc4*[*simp*]:

readLocs (*Config 4* (*State* (*Vstore vs*) *avst (Heap h) p*)) =
 (*let i = array-loc aa1 (nat (vs xx)) avst;*
*j = array-loc aa2 (nat ((h i) * 512)) avst)*
in {*i, j*}

$\langle proof \rangle$

lemma *nextB-pc5[simp]*:

$nextB (Config\ 5\ s,\ ibT,\ ibUT) = (Config\ 6\ s,\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc5[simp]*:

$readLocs (Config\ 5\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p)) =$
 $\{\}$
 $\langle proof \rangle$

lemma *nextB-stepB-pc*:

$pc < 6 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_B nextB (Config\ pc\ s,\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *not-finalB*:

$pc < 6 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s,\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *finalB-pc-iff'*:

$pc < 6 \implies$
 $finalB (Config\ pc\ s,\ ibT,\ ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil)$
 $\langle proof \rangle$

lemma *finalB-pc-iff*:

$pc \leq 6 \implies$
 $finalB (Config\ pc\ s,\ ibT,\ ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil \vee pc = 6)$
 $\langle proof \rangle$

lemma *finalB-pcOf-iff[simp]*:

$pcOf\ cfg \leq 6 \implies$
 $finalB (cfg,\ ibT,\ ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = 6)$
 $\langle proof \rangle$

definition *vs_i-t cfg* $\equiv (vstore (getVstore (stateOf\ cfg))\ xx) < NN$

definition *vs_i-f cfg* $\equiv (vstore (getVstore (stateOf\ cfg))\ xx) \geq NN$

lemma *vs-xx-cases*: $vs_i\text{-}t\ cfg \vee vs_i\text{-}f\ cfg$ $\langle proof \rangle$

lemmas $vs_i\text{-defs} = vs_i\text{-t-def } vs_i\text{-f-def}$

lemma $bool\text{-invar}[simp]: \neg vs_i\text{-t} (Config\ 6\ s) \implies vs_i\text{-t} (Config\ 6\ s) \implies (Config\ 6\ s, ib1) \rightarrow B (Config\ 6\ s, ib1) \implies False$
<proof>

lemma $nextB\text{-vs-consistent-aux}$:

$2 \leq pc \wedge pc < 6 \implies$
 $(nextB (Config\ pc (State (Vstore\ vs) avst (Heap\ h) p), ibT, ibUT)) = (Config\ pc'$
 $(State (Vstore\ vs') avst'' (Heap\ h') p'), ibT', ibUT')) \implies$
 $avst = avst'' \wedge$
 $vs\ xx = vs'\ xx \wedge$
 $h = h' \wedge$
 $pc < pc'$
<proof>

lemma $nextB\text{-vs-consistent}$:

$2 \leq pcOf\ cfg \wedge pcOf\ cfg < 6 \implies$
 $(nextB (cfg, ibT, ibUT)) = (cfg', ibT', ibUT') \implies$
 $(getAvstore (stateOf\ cfg)) = (getAvstore (stateOf\ cfg')) \wedge$
 $(getHheap (stateOf\ cfg)) = (getHheap (stateOf\ cfg')) \wedge$
 $vstore (getVstore (stateOf\ cfg))\ xx = vstore (getVstore (stateOf\ cfg'))\ xx$
<proof>

lemma $nextB\text{-vs}_i\text{-t-consistent}$:

$2 \leq pcOf\ cfg \wedge pcOf\ cfg < 6 \implies$
 $(nextB (cfg, ibT, ibUT)) = (cfg', ibT', ibUT') \implies$
 $vs_i\text{-t}\ cfg \longleftrightarrow vs_i\text{-t}\ cfg'$
<proof>

lemma $nextB\text{-vs}_i\text{-f-consistent}$:

$2 \leq pcOf\ cfg \wedge pcOf\ cfg < 6 \implies$
 $(nextB (cfg, ibT, ibUT)) = (cfg', ibT', ibUT') \implies$
 $vs_i\text{-f}\ cfg \longleftrightarrow vs_i\text{-f}\ cfg'$
<proof>

end

8.2 Proof

theory $Fun1\text{-insecure}$
imports $Fun1$
begin

8.2.1 Concrete leak

definition $PC \equiv \{0..6\}$

definition $same\text{-xx}\ cfg3\ cfgs3\ cfg4\ cfgs4 \equiv$

$vstore (getVstore (stateOf\ cfg3))\ xx = vstore (getVstore (stateOf\ cfg4))\ xx \wedge$
 $(\forall\ cfg3' \in set\ cfgs3. vstore (getVstore (stateOf\ cfg3'))\ xx = vstore (getVstore (stateOf\ cfg3))\ xx) \wedge$
 $(\forall\ cfg4' \in set\ cfgs4. vstore (getVstore (stateOf\ cfg4'))\ xx = vstore (getVstore (stateOf\ cfg4))\ xx)$

definition $trueProg = \{2,3,4,5,6\}$

definition $falseProg = \{2,3,5,6\}$

definition $pstate_1 \equiv update\ pstate_0\ [3]$

definition $pstate_2 \equiv update\ pstate_1\ [5,5]$

lemmas $pstate-def = pstate_1-def\ pstate_2-def$

fun $hh_3 :: nat \Rightarrow int$ **where**

$hh_3\ x = (if\ x = (nat\ NN + 1)\ then\ 5\ else\ 0)$

definition $h_3 \equiv (Heap\ hh_3)$

fun $hh_4 :: nat \Rightarrow int$ **where**

$hh_4\ x = (if\ x = (nat\ NN + 1)\ then\ 6\ else\ 0)$

definition $h_4 \equiv (Heap\ hh_4)$

lemmas $h-def = h_3-def\ h_4-def\ hh_3.simps\ hh_4.simps$

lemma $ss-neq-aux1 : nat(5 * 512) \neq nat(6 * 512)$ *<proof>*

lemma $ss-neq-aux2 : nat(3 * 512) \neq nat(5 * 512)$ *<proof>*

lemmas $ss-neq = ss-neq-aux1\ ss-neq-aux2$

definition $p \equiv nat\ size-aa1 + nat\ size-aa2$

definition $vs_1 \equiv (vs_0(x := NN + 1))$

definition $vs_2 \equiv (vs_1(tt := 0))$

definition $aa1_i \equiv array-loc\ aa1\ (nat\ (vs_2\ xx))\ avst'$

definition $aa2_{vs3} \equiv \text{array-loc } aa2 \text{ (nat (hh}_3 \text{ aa1}_i * 512)) \text{ avst}'$

definition $vs_{33} = vs_2(tt := hh_3 \text{ aa2}_{vs3})$

definition $aa2_{vs4} \equiv \text{array-loc } aa2 \text{ (nat (hh}_4 \text{ aa1}_i * 512)) \text{ avst}'$

definition $vs_{34} = vs_2(tt := hh_4 \text{ aa2}_{vs4})$

lemmas $reads_m\text{-def} = aa1_i\text{-def } aa2_{vs3}\text{-def } aa2_{vs4}\text{-def}$

lemmas $vs\text{-def} = vs_0.\text{sims } vs_1\text{-def } vs_2\text{-def } vs_{33}\text{-def } vs_{34}\text{-def}$

definition $s_{03} \equiv (\text{State (Vstore } vs_0) \text{ avst}' h_3 p)$

definition $s_{13} \equiv (\text{State (Vstore } vs_1) \text{ avst}' h_3 p)$

definition $s_{23} \equiv (\text{State (Vstore } vs_2) \text{ avst}' h_3 p)$

definition $s_{33} \equiv (\text{State (Vstore } vs_{33}) \text{ avst}' h_3 p)$

definition $s_{04} \equiv (\text{State (Vstore } vs_0) \text{ avst}' h_4 p)$

definition $s_{14} \equiv (\text{State (Vstore } vs_1) \text{ avst}' h_4 p)$

definition $s_{24} \equiv (\text{State (Vstore } vs_2) \text{ avst}' h_4 p)$

definition $s_{34} \equiv (\text{State (Vstore } vs_{34}) \text{ avst}' h_4 p)$

lemmas $s\text{-def} = s_{03}\text{-def } s_{13}\text{-def } s_{23}\text{-def } s_{33}\text{-def}$

$s_{04}\text{-def } s_{14}\text{-def } s_{24}\text{-def } s_{34}\text{-def}$

definition $(s_{30}:: \text{stateO}) \equiv (pstate_0, (\text{Config } 0 \text{ } s_{03}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s_{31}:: \text{stateO}) \equiv (pstate_0, (\text{Config } 1 \text{ } s_{03}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s_{32}:: \text{stateO}) \equiv (pstate_0, (\text{Config } 2 \text{ } s_{13}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s_{33}:: \text{stateO}) \equiv (pstate_0, (\text{Config } 3 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s_{34}:: \text{stateO}) \equiv (pstate_1, (\text{Config } 5 \text{ } s_{23}), [\text{Config } 4 \text{ } s_{23}], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s_{35}:: \text{stateO}) \equiv (pstate_1, (\text{Config } 5 \text{ } s_{23}), [\text{Config } 5 \text{ } s_{33}], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

definition $(s_{36}:: \text{stateO}) \equiv (pstate_2, (\text{Config } 5 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

definition $(s_{37}:: \text{stateO}) \equiv (pstate_2, (\text{Config } 6 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

lemmas $s3\text{-def} = s_{30}\text{-def } s_{31}\text{-def } s_{32}\text{-def } s_{33}\text{-def } s_{34}\text{-def } s_{35}\text{-def } s_{36}\text{-def } s_{37}\text{-def}$

lemmas $state\text{-def} = s\text{-def } h\text{-def } vs\text{-def } reads_m\text{-def } pstate\text{-def } avst\text{-defs}$

definition $s\beta\text{-trans} \equiv [s\beta_0, s\beta_1, s\beta_2, s\beta_3, s\beta_4, s\beta_5, s\beta_6, s\beta_7]$
lemmas $s\beta\text{-trans-defs} = s\beta\text{-trans-def } s\beta\text{-def}$

lemma $hd\text{-}s\beta\text{-trans}[simp]: hd\ s\beta\text{-trans} = s\beta_0 \langle proof \rangle$
lemma $s\beta\text{-trans-nemp}[simp]: s\beta\text{-trans} \neq [] \langle proof \rangle$

lemma $s\beta_{01}[simp]: s\beta_0 \rightarrow S\ s\beta_1$
 $\langle proof \rangle$

lemma $s\beta_{12}[simp]: s\beta_1 \rightarrow S\ s\beta_2$
 $\langle proof \rangle$

lemma $s\beta_{23}[simp]: s\beta_2 \rightarrow S\ s\beta_3$
 $\langle proof \rangle$

lemma $s\beta_{34}[simp]: s\beta_3 \rightarrow S\ s\beta_4$
 $\langle proof \rangle$

lemma $s\beta_{45}[simp]: s\beta_4 \rightarrow S\ s\beta_5$
 $\langle proof \rangle$

lemma $s\beta_{56}[simp]: s\beta_5 \rightarrow S\ s\beta_6$
 $\langle proof \rangle$

lemma $s\beta_{67}[simp]: s\beta_6 \rightarrow S\ s\beta_7$
 $\langle proof \rangle$

lemma $finalS\text{-}s\beta_7[simp]: finalS\ s\beta_7$
 $\langle proof \rangle$

lemmas $s\beta\text{-trans-simps} = s\beta_{01} s\beta_{12} s\beta_{23} s\beta_{34} s\beta_{45} s\beta_{56} s\beta_{67}$

definition $(s4_0:: stateO) \equiv (pstate_0, (Config\ 0\ s_{04}), [], repeat\ (NN+1), repeat\ (NN+1), \{\})$

definition $(s4_1:: stateO) \equiv (pstate_0, (Config\ 1\ s_{04}), [], repeat\ (NN+1), repeat\ (NN+1), \{\})$

definition $(s4_2:: stateO) \equiv (pstate_0, (Config\ 2\ s_{14}), [], repeat\ (NN+1), repeat\ (NN+1), \{\})$

definition $(s4_3:: stateO) \equiv (pstate_0, (Config\ 3\ s_{24}), [], repeat\ (NN+1), repeat\ (NN+1), \{\})$

definition $(s4_4:: stateO) \equiv (pstate_1, (Config\ 5\ s_{24}), [Config\ 4\ s_{24}], repeat\ (NN+1), repeat\ (NN+1), \{\})$

definition $(s4_5:: stateO) \equiv (pstate_1, (Config\ 5\ s_{24}), [Config\ 5\ s_{34}], repeat\ (NN+1), repeat\ (NN+1), \{aa2_{vs4}, aa1_i\})$

definition ($s4_6:: stateO$) \equiv ($pstate_2$, ($Config\ 5\ s_{24}$), [], $repeat\ (NN+1)$, $repeat\ (NN+1)$, $\{aa2_{vs4}, aa1_i\}$)

definition ($s4_7:: stateO$) \equiv ($pstate_2$, ($Config\ 6\ s_{24}$), [], $repeat\ (NN+1)$, $repeat\ (NN+1)$, $\{aa2_{vs4}, aa1_i\}$)

lemmas $s4-def = s4_0-def\ s4_1-def\ s4_2-def\ s4_3-def\ s4_4-def\ s4_5-def\ s4_6-def\ s4_7-def$

definition $s4-trans \equiv [s4_0, s4_1, s4_2, s4_3, s4_4, s4_5, s4_6, s4_7]$

lemmas $s4-trans-defs = s4-trans-def\ s4-def$

lemma $hd-s4-trans[simp]: hd\ s4-trans = s4_0\ \langle proof \rangle$

lemma $s4-trans-nemp[simp]: s4-trans \neq []\ \langle proof \rangle$

lemma $s4_{01}[simp]: s4_0 \rightarrow S\ s4_1$
 $\langle proof \rangle$

lemma $s4_{12}[simp]: s4_1 \rightarrow S\ s4_2$
 $\langle proof \rangle$

lemma $s4_{24}[simp]: s4_2 \rightarrow S\ s4_3$
 $\langle proof \rangle$

lemma $s4_{34}[simp]: s4_3 \rightarrow S\ s4_4$
 $\langle proof \rangle$

lemma $s4_{45}[simp]: s4_4 \rightarrow S\ s4_5$
 $\langle proof \rangle$

lemma $s4_{56}[simp]: s4_5 \rightarrow S\ s4_6$
 $\langle proof \rangle$

lemma $s4_{67}[simp]: s4_6 \rightarrow S\ s4_7$
 $\langle proof \rangle$

lemma $finalS-s4_7[simp]: finalS\ s4_7$
 $\langle proof \rangle$

lemmas $s4-trans-simps = s4_{01}\ s4_{12}\ s4_{24}\ s4_{34}\ s4_{45}\ s4_{56}\ s4_{67}$

8.2.2 Auxillary lemmas for disproof

lemma $validS-s3-trans[simp]: Opt.validS\ s3-trans$
 $\langle proof \rangle$

lemma *validS-s4-trans*[simp]: *Opt.validS s4-trans*
 ⟨proof⟩

lemma *finalS-s3*[simp]: *finalS (last s3-trans)* ⟨proof⟩
lemma *finalS-s4*[simp]: *finalS (last s4-trans)* ⟨proof⟩

lemma *filter-s3*[simp]: (*filter isIntO (butlast s3-trans)*) = (*butlast s3-trans*)
 ⟨proof⟩

lemma *filter-s4*[simp]: (*filter isIntO (butlast s4-trans)*) = (*butlast s4-trans*)
 ⟨proof⟩

lemma *S-s3-trans*[simp]: *Opt.S s3-trans* = [*s03*]
 ⟨proof⟩

lemma *S-s4-trans*[simp]: *Opt.S s4-trans* = [*s04*]
 ⟨proof⟩

lemma *finalB-noStep*[simp]: $\bigwedge s1'. \text{finalB } (cfg1, ibT1, ibUT1) \implies (cfg1, ibT1, ibUT1, ls1) \rightarrow N s1' \implies \text{False}$
 ⟨proof⟩

8.2.3 Disproof of fun1

fun *common-memory*::*config* \Rightarrow *config* \Rightarrow *bool* **where**
common-memory *cfg1* *cfg2* =
 (let *h1* = (*getHheap (stateOf cfg1)*);
 h2 = (*getHheap (stateOf cfg2)*) in
 ($\forall x \in \text{read-add. } h1\ x = h2\ x \wedge h1\ x = 0$) \wedge
 (*getAvstore (stateOf cfg1)*) = *avst'* \wedge
 (*getAvstore (stateOf cfg2)*) = *avst'*)

lemma *heap-eq0*[simp]: $\forall x. x \neq \text{Suc NN} \longrightarrow hh1'\ x = hh2'\ x \wedge hh1'\ x = 0 \implies hh2'\ NN = 0$
 ⟨proof⟩

lemma *heap1-eq0*[simp]: $\forall x. x \neq \text{Suc NN} \longrightarrow hh1'\ x = hh2'\ x \wedge hh1'\ x = 0 \implies vs2\ xx < NN \implies hh2'\ (\text{nat } (vs2\ xx)) = 0$
 ⟨proof⟩

fun Γ -*inv*::*stateV* \Rightarrow *state list* \Rightarrow *stateV* \Rightarrow *state list* \Rightarrow *bool* **where**
 Γ -*inv* (*cfg1*, *ibT1*, *ibUT1*, *ls1*) *sl1* (*cfg2*, *ibT2*, *ibUT2*, *ls2*) *sl2* =
 (
 (*pcOf* *cfg1* = *pcOf* *cfg2*) \wedge
 (*pcOf* *cfg1* < 2 \longrightarrow *ibUT1* \neq *LNil* \wedge *ibUT2* \neq *LNil*) \wedge

$(pcOf\ cfg1 > 2 \longrightarrow same\text{-}var\text{-}val\ tt\ 0\ cfg1\ cfg2) \wedge$
 $(pcOf\ cfg1 > 1 \longrightarrow (same\text{-}var\ xx\ cfg1\ cfg2) \wedge$
 $(vs_i\text{-}t\ cfg1 \longrightarrow pcOf\ cfg1 \in trueProg) \wedge$
 $(vs_i\text{-}f\ cfg1 \longrightarrow pcOf\ cfg1 \in falseProg))$
 \wedge
 $ls1 = ls2 \wedge$
 $pcOf\ cfg1 \in PC \wedge$
 $common\text{-}memory\ cfg1\ cfg2$
 $)$

declare $\Gamma\text{-}inv.simps[simp\ del]$
lemmas $\Gamma\text{-}def = \Gamma\text{-}inv.simps$
lemmas $\Gamma\text{-}defs = \Gamma\text{-}def\ common\text{-}memory.simps\ PC\text{-}def\ aa1_i\text{-}def$
 $trueProg\text{-}def\ falseProg\text{-}def\ same\text{-}var\text{-}val\text{-}def\ same\text{-}var\text{-}def$

lemma $\Gamma\text{-}implies:\Gamma\text{-}inv\ (cfg1,ibT1,ibUT1,ls1)\ sl1\ (cfg2,ibT2,ibUT2,ls2)\ sl2 \implies$
 $pcOf\ cfg1 \leq 6 \wedge pcOf\ cfg2 \leq 6 \wedge$
 $(pcOf\ cfg1 = 4 \longrightarrow vs_i\text{-}t\ cfg1) \wedge$
 $(pcOf\ cfg2 = 4 \longrightarrow vs_i\text{-}t\ cfg2) \wedge$
 $(pcOf\ cfg1 > 1 \longrightarrow vs_i\text{-}t\ cfg1 \longleftrightarrow vs_i\text{-}t\ cfg2) \wedge$
 $(finalB\ (cfg1,ibT1,ibUT1) \longleftrightarrow pcOf\ cfg1 = 6) \wedge$
 $(finalB\ (cfg2,ibT2,ibUT2) \longleftrightarrow pcOf\ cfg2 = 6)$
 $\langle proof \rangle$

lemma $istateO\text{-}s3[simp]:istateO\ s3_0 \langle proof \rangle$
lemma $istateO\text{-}s4[simp]:istateO\ s4_0 \langle proof \rangle$

lemma $validFromS\text{-}s3[simp]:Opt.validFromS\ s3_0\ s3\text{-}trans$
 $\langle proof \rangle$

lemma $validFromS\text{-}s4[simp]:Opt.validFromS\ s4_0\ s4\text{-}trans$
 $\langle proof \rangle$

lemma $completedFromO\text{-}s3[simp]:completedFromO\ s3_0\ s3\text{-}trans$
 $\langle proof \rangle$

lemma *completedFromO-s4[simp]:completedFromO s4_0 s4-trans*
⟨proof⟩

lemma *Act-eq[simp]:Opt.A s3-trans = Opt.A s4-trans*
⟨proof⟩

lemma *aa2-neq:aa2_vs3 ≠ aa2_vs4*
⟨proof⟩

lemma *aa1-neq:aa2_vs3 ≠ aa1_i*
⟨proof⟩

lemma *aa1-neq2:aa2_vs4 ≠ aa1_i*
⟨proof⟩

lemma *Obs-neq[simp]:Opt.O s3-trans ≠ Opt.O s4-trans*
⟨proof⟩

lemma $\Gamma\text{-init}[simp]: \wedge s1\ s2. \text{istateV } s1 \implies \text{corrState } s1\ s3_0 \implies \text{istateV } s2 \implies$
 $\text{corrState } s2\ s4_0 \implies \Gamma\text{-inv } s1\ [s_{03}]\ s2\ [s_{04}]$
⟨proof⟩

lemma *val-neq-1:nat (hh2' (nat (vs2 xx)) * 512) ≠ 1*
⟨proof⟩

lemma *unwindSD[simp]:Rel-Sec.unwindSDCond validTransV istateV isSecV get-*
SecV isIntV getIntV Γ -inv
⟨proof⟩

theorem $\neg\text{rsecure}$
⟨proof⟩

end

9 Proof of Relative Security for fun2

theory *Fun2*
imports
../Instance-IMP/Instance-Secret-IMem
Relative-Security.Unwinding-fin
begin

9.1 Function definition and Boilerplate

no-notation *bot* (\perp)

consts *NN* :: *nat*

lemma *NN*: $NN \geq 0$ *<proof>*

definition *aa1* :: *avname* **where** *aa1* = "a1"

definition *aa2* :: *avname* **where** *aa2* = "a2"

definition *xx* :: *avname* **where** *xx* = "xx"

definition *tt* :: *avname* **where** *tt* = "tt"

lemmas *vvars-defs* = *aa1-def aa2-def xx-def tt-def*

lemma *vvars-dff*[*simp*]:

aa1 \neq *aa2* *aa1* \neq *xx* *aa1* \neq *tt*

aa2 \neq *aa1* *aa2* \neq *xx* *aa2* \neq *tt*

xx \neq *aa1* *xx* \neq *aa2* *xx* \neq *tt*

tt \neq *aa1* *tt* \neq *aa2* *tt* \neq *xx*

<proof>

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

lemma *aa1*: $size-aa1 \geq 0$ **and** *aa2*: $size-aa2 \geq 0$ *<proof>*

fun *initAvstore* :: *avstore* \Rightarrow *bool* **where**

initAvstore (*Avstore as*) = (*as aa1* = (0, *nat size-aa1*) \wedge *as aa2* = (*nat size-aa1*, *nat size-aa2*))

fun *istate* :: *state* \Rightarrow *bool* **where**

istate s = (*initAvstore* (*getAvstore s*))

definition *prog* \equiv

```
[
  / Start ,
  / Input U xx ,
  / tt ::= (N 0) ,
  / IfJump (Less (V xx) (N NN)) 4 6 ,
  / Fence ,
  / tt ::= (VA aa2 (Times (VA aa1 (V xx)) (N 512))),
  / Output U (V tt)
]
```

lemma *cases-6*: (*i::pcounter*) = 0 \vee *i* = 1 \vee *i* = 2 \vee *i* = 3 \vee *i* = 4 \vee *i* = 5 \vee

$i = 6 \vee i > 6$
 $\langle proof \rangle$

lemma *xx-NN-cases*: $vs\ xx < int(NN) \vee vs\ xx \geq int(NN)$ $\langle proof \rangle$

lemma *is-If-pcOf[simp]*:
 $pcOf\ cfg < 6 \implies is-IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 3$
 $\langle proof \rangle$

lemma *is-If-pc[simp]*:
 $pc < 6 \implies is-IfJump\ (prog\ !\ pc) \longleftrightarrow pc = 3$
 $\langle proof \rangle$

lemma *eq-Fence-pc[simp]*:
 $pc < 6 \implies prog\ !\ pc = Fence \longleftrightarrow pc = 4$
 $\langle proof \rangle$

consts *mispred* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$
consts *resolve* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$

consts *update* :: $predState \Rightarrow pcounter\ list \Rightarrow predState$
consts *initPstate* :: $predState$

interpretation *Prog-Mispred-Init* **where**
 $prog = prog$ **and** $initPstate = initPstate$ **and**
 $mispred = mispred$ **and** $resolve = resolve$ **and** $update = update$ **and**
 $istate = istate$
 $\langle proof \rangle$

abbreviation

$stepB\ abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
 $bool$ (**infix** $\langle \rightarrow B \rangle$ 55)
where $x \rightarrow B y == stepB\ x\ y$

abbreviation

$stepsB\ abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
 $bool$ (**infix** $\langle \rightarrow B^* \rangle$ 55)
where $x \rightarrow B^* y == star\ stepB\ x\ y$

abbreviation

$stepM\ abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$

bool (**infix** $\langle \rightarrow M \rangle$ 55)
where $x \rightarrow M y == stepM x y$

abbreviation

stepN-abbrev :: *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *bool* (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == stepN x y$

abbreviation

stepsN-abbrev :: *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *bool* (**infix** $\langle \rightarrow N^* \rangle$ 55)
where $x \rightarrow N^* y == star stepN x y$

abbreviation

stepS-abbrev :: *configS* \Rightarrow *configS* \Rightarrow *bool* (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == stepS x y$

abbreviation

stepsS-abbrev :: *configS* \Rightarrow *configS* \Rightarrow *bool* (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == star stepS x y$

lemma *endPC[simp]*: *endPC* = 7
 $\langle proof \rangle$

lemma *is-getUntrustedInput-pcOf[simp]*: *pcOf cfg* < 6 \implies *is-getInput* (*prog!*(*pcOf cfg*)) \longleftrightarrow *pcOf cfg* = 1
 $\langle proof \rangle$

lemma *start[simp]*: *prog* ! 0 = *Start*
 $\langle proof \rangle$

lemma *getUntrustedInput-pcOf[simp]*: *prog* ! 1 = *Input U xx*
 $\langle proof \rangle$

lemma *if-stat[simp]*: *prog* ! 3 = (*IfJump* (*Less* (*V xx*) (*N NN*)) 4 6)
 $\langle proof \rangle$

lemma *isOutput1[simp]*: *prog* ! 6 = *Output U* (*V tt*)
 $\langle proof \rangle$

lemma *is-Output-pcOf[simp]*: *pcOf cfg* < 7 \implies *is-Output* (*prog!*(*pcOf cfg*)) \longleftrightarrow *pcOf cfg* = 6
 $\langle proof \rangle$

lemma *is-Fence-pcOf[simp]*: $pcOf\ cf g < 7 \implies (prog!(pcOf\ cf g)) = Fence \longleftrightarrow pcOf\ cf g = 4$
 ⟨proof⟩

lemma *is-Output[simp]*: $is-Output\ (prog\ !\ 6)$
 ⟨proof⟩

lemma *isSecV-pcOf[simp]*:
 $isSecV\ (cfg, ibT, ibUT) \longleftrightarrow pcOf\ cf g = 0$
 ⟨proof⟩

lemma *isSecO-pcOf[simp]*:
 $isSecO\ (pstate, cfg, cfgs, ibT, ibUT, ls) \longleftrightarrow (pcOf\ cf g = 0 \wedge cfgs = [])$
 ⟨proof⟩

lemma *getInputT-not[simp]*: $pcOf\ cf g < 7 \implies (prog\ !\ pcOf\ cf g) \neq Input\ T\ inp$
 ⟨proof⟩

lemma *getActV-pcOf[simp]*:
 $pcOf\ cf g < 7 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cf g = 1\ then\ lhd\ ibUT\ else\ \perp)$
 ⟨proof⟩

lemma *getObsV-pcOf[simp]*:
 $pcOf\ cf g < 7 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cf g = 6\ then$
 $(outOf\ (prog!(pcOf\ cf g))\ (stateOf\ cf g),\ ls)$
 $else\ \perp$
 $)$
 ⟨proof⟩

lemma *getActO-pcOf[simp]*:
 $pcOf\ cf g < 7 \implies$
 $getActO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 $(if\ pcOf\ cf g = 1 \wedge cfgs = []\ then\ lhd\ ibUT\ else\ \perp)$
 ⟨proof⟩

lemma *getObsO-pcOf[simp]*:

$pcOf\ cfg < 7 \implies$
 $getObsO\ (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 6 \wedge cfs = [])\ then$
 $\ (outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg), ls)$
 $\ else\ \perp$
 $)$
 $\langle proof \rangle$

lemma $eqSec\text{-}pc0[simp]$:
 $eqSec\ (cfg1, ibT, ibUT1, ls1)\ (pstate3, cfg3, cfs3, ibT, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
 $\langle proof \rangle$

lemma $nextB\text{-}pc0[simp]$:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $nextB\text{-}pc0'[simp]$: $nextB\ (Config\ 0\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
 $=$
 $(Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs\text{-}pc0[simp]$:
 $readLocs\ (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB\text{-}pc1[simp]$:
 $ibUT \neq LNil \implies nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs\ (xx := lhd\ ibUT))))\ avst\ h\ p, ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma $readLocs\text{-}pc1[simp]$:
 $readLocs\ (Config\ 1\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB\text{-}pc1'[simp]$:
 $ibUT \neq LNil \implies nextB\ (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$

=
 (Config 2 (State (Vstore (vs(xx := lhd ibUT))) avst h p), ibT, ltl ibUT)
 ⟨proof⟩

lemma readLocs-pc1 [simp]:
 readLocs (Config (Suc 0) s) = {}
 ⟨proof⟩

lemma nextB-pc2 [simp]:
 nextB (Config 2 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config 3 (State (Vstore (vs(tt := 0))) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma readLocs-pc2 [simp]:
 readLocs (Config 2 s) = {}
 ⟨proof⟩

lemma nextB-pc3-then [simp]:
 vs xx < NN \implies
 nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config 4 (State (Vstore vs) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextB-pc3-else [simp]:
 vs xx \geq NN \implies
 nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config 6 (State (Vstore vs) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextB-pc3:
 nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config (if vs xx < NN then 4 else 6) (State (Vstore vs) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextM-pc3-then [simp]:
 vs xx \geq NN \implies
 nextM (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config 4 (State (Vstore vs) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma nextM-pc3-else [simp]:
 vs xx < NN \implies
 nextM (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =
 (Config 6 (State (Vstore vs) avst h p), ibT, ibUT)
 ⟨proof⟩

lemma *nextM-pc3*:

nextM (*Config 3* (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*) =
(*Config* (*if vs xx < NN then 6 else 4*) (*State* (*Vstore vs*) *avst h p*), *ibT*, *ibUT*)
<proof>

lemma *readLocs-pc3[simp]*:

readLocs (*Config 3 s*) = {}
<proof>

lemma *nextB-pc4[simp]*:

nextB (*Config 4 s*, *ibT*, *ibUT*) = (*Config 5 s*, *ibT*, *ibUT*)
<proof>

lemma *readLocs-pc4[simp]*:

readLocs (*Config 4 s*) = {}
<proof>

lemma *nextB-pc5[simp]*:

nextB (*Config 5* (*State* (*Vstore vs*) *avst (Heap h) p*), *ibT*, *ibUT*) =
(*let l = (array-loc aa2 (nat (h (array-loc aa1 (nat (vs xx)) avst) * 512)) avst)*
in (Config 6 (State (Vstore (vs(tt := h l))) avst (Heap h) p)), *ibT*, *ibUT*)
<proof>

lemma *readLocs-pc5[simp]*:

readLocs (*Config 5 (State (Vstore vs) avst (Heap h) p)*) =
{*array-loc aa2 (nat (h (array-loc aa1 (nat (vs xx)) avst) * 512)) avst*, *array-loc*
aa1 (nat (vs xx)) avst}
<proof>

lemma *nextB-pc6[simp]*:

nextB (*Config 6 s*, *ibT*, *ibUT*) = (*Config 7 s*, *ibT*, *ibUT*)
<proof>

lemma *readLocs-pc6[simp]*:

readLocs (*Config 6 (State (Vstore vs) avst (Heap h) p)*) =
{}
<proof>

lemma *nextB-stepB-pc*:

$pc < 7 \implies (pc = 1 \implies ibUT \neq LNil) \implies$

$(\text{Config } pc \ s, \text{ ibT}, \text{ ibUT}) \rightarrow_B \text{nextB } (\text{Config } pc \ s, \text{ ibT}, \text{ ibUT})$
 ⟨proof⟩

lemma *not-finalB*:

$pc < 7 \implies (pc = 1 \implies \text{ibUT} \neq \text{LNil}) \implies$
 $\neg \text{finalB } (\text{Config } pc \ s, \text{ ibT}, \text{ ibUT})$
 ⟨proof⟩

lemma *finalB-pc-iff'*:

$pc < 7 \implies$
 $\text{finalB } (\text{Config } pc \ s, \text{ ibT}, \text{ ibUT}) \longleftrightarrow$
 $(pc = 1 \wedge \text{ibUT} = \text{LNil})$
 ⟨proof⟩

lemma *finalB-pc-iff*:

$pc \leq 7 \implies$
 $\text{finalB } (\text{Config } pc \ s, \text{ ibT}, \text{ ibUT}) \longleftrightarrow$
 $(pc = 1 \wedge \text{ibUT} = \text{LNil} \vee pc = 7)$
 ⟨proof⟩

lemma *finalB-pcOf-iff[simp]*:

$pcOf \text{ cfg} \leq 7 \implies$
 $\text{finalB } (\text{cfg}, \text{ ibT}, \text{ ibUT}) \longleftrightarrow (pcOf \ \text{cfg} = 1 \wedge \text{ibUT} = \text{LNil} \vee pcOf \ \text{cfg} = 7)$
 ⟨proof⟩

lemma *finalS-cond:pcOf cfg < 7 \implies cfgs = [] \implies (pcOf cfg = 1 \implies ibUT \neq LNil) \implies \neg finalS (pstate, cfg, cfgs, ibT, ibUT, ls)*
 ⟨proof⟩

lemma *finalS-cond-spec*:

$pcOf \ \text{cfg} < 7 \implies$
 $(pcOf \ (\text{last } \text{cfgs}) = 4 \wedge pcOf \ \text{cfg} = 6) \vee (pcOf \ (\text{last } \text{cfgs}) = 6 \wedge pcOf \ \text{cfg} =$
 $4) \implies$
 $\text{length } \text{cfgs} = \text{Suc } 0 \implies$
 $\neg \text{finalS } (\text{pstate}, \text{ cfg}, \text{ cfgs}, \text{ ibT}, \text{ ibUT}, \text{ ls})$
 ⟨proof⟩

end

9.2 Proof

theory *Fun2-secure*
imports *Fun2*

begin

definition $PC \equiv \{0..6\}$

definition $same\text{-}xx\ cfg3\ cfs3\ cfg4\ cfs4 \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg3))\ xx = vstore\ (getVstore\ (stateOf\ cfg4))\ xx \wedge$
 $(\forall\ cfg3' \in set\ cfs3.\ vstore\ (getVstore\ (stateOf\ cfg3'))\ xx = vstore\ (getVstore\ (stateOf\ cfg3))\ xx) \wedge$
 $(\forall\ cfg4' \in set\ cfs4.\ vstore\ (getVstore\ (stateOf\ cfg4'))\ xx = vstore\ (getVstore\ (stateOf\ cfg4))\ xx)$

definition $beforeInput = \{0,1\}$

definition $afterInput = \{2,3,4,5,6\}$

definition $inThenBranch = \{4,5,6\}$

definition $startOfThenBranch = 4$

definition $elseBranch = 6$

definition $common :: stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$

where

$common = (\lambda$
 $\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $\ (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $\ statA$
 $\ (cfg1, ibT1, ibUT1, ls1)$
 $\ (cfg2, ibT2, ibUT2, ls2)$
 $\ statO.$
 $(pstate3 = pstate4 \wedge$
 $cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge$
 $pcOf\ cfg3 = pcOf\ cfg4 \wedge map\ pcOf\ cfs3 = map\ pcOf\ cfs4 \wedge$
 $pcOf\ cfg3 \in PC \wedge pcOf\ (set\ cfs3) \subseteq PC \wedge$
 $///$
 $array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3)) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfs3.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3')) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfs4.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4')) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4))) \wedge$
 $array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3)) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfs3.\ array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3')) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfs4.\ array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4')) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4))) \wedge$

///
 (statA = Diff \longrightarrow statO = Diff)))

lemma *common-implies: common* (pstate3, cfg3, cfs3, ibT, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT, ibUT4, ls4)
 statA
 (cfg1, ibT, ibUT1, ls1)
 (cfg2, ibT, ibUT2, ls2)
 statO \implies
 pcOf cfg1 < 8 \wedge pcOf cfg2 = pcOf cfg1
 <proof>

definition $\Delta 0 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

$\Delta 0 = (\lambda \text{num}$
 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (common (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge
 ibUT1 = ibUT3 \wedge ibUT2 = ibUT4 \wedge
 (pcOf cfg3 > 1 \longrightarrow same-xx cfg3 cfs3 cfg4 cfs4) \wedge
 (pcOf cfg3 < 2 \longrightarrow ibUT1 \neq LNil \wedge ibUT2 \neq LNil \wedge ibUT3 \neq LNil \wedge ibUT4 \neq LNil)
 \wedge
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 pcOf cfg3 \in beforeInput \wedge
 noMisSpec cfs3
))

lemmas $\Delta 0\text{-defs} = \Delta 0\text{-def}$ *common-def* *PC-def*
beforeInput-def
same-xx-def *noMisSpec-def*

lemma $\Delta 0\text{-implies: } \Delta 0 \text{ num}$
 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 (pcOf cfg3 = 1 \longrightarrow ibUT3 \neq LNil) \wedge

$(pcOf\ cfg4 = 1 \longrightarrow ibUT4 \neq LNil) \wedge$
 $pcOf\ cfg1 < 7 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $cfgs3 = [] \wedge pcOf\ cfg3 < 7 \wedge$
 $cfgs4 = [] \wedge pcOf\ cfg4 < 7$
 ⟨proof⟩

definition $\Delta 1 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 1 = (\lambda\ num$
 $(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $ls1 = ls3 \wedge ls2 = ls4 \wedge$
 $same-xx\ cfg3\ cfgs3\ cfg4\ cfgs4 \wedge$
 $pcOf\ cfg3 \in afterInput \wedge$
 $noMisSpec\ cfgs3$
 $))$

lemmas $\Delta 1-defs = \Delta 1-def\ common-def\ PC-def\ afterInput-def\ noMisSpec-def\ same-xx-def$

lemma $\Delta 1$ -implies: $\Delta 1\ num$

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \Longrightarrow$
 $pcOf\ cfg1 < 7 \wedge$
 $cfgs3 = [] \wedge pcOf\ cfg3 \neq 1 \wedge pcOf\ cfg3 < 7 \wedge$
 $cfgs4 = [] \wedge pcOf\ cfg4 \neq 1 \wedge pcOf\ cfg4 < 7$
 ⟨proof⟩

definition $\Delta 2 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 2 = (\lambda\ num$
 $(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$

```

      (cfg1,ibT1,ibUT1,ls1)
      (cfg2,ibT2,ibUT2,ls2)
      statO.
    (common (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
      (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
      statA
      (cfg1,ibT1,ibUT1,ls1)
      (cfg2,ibT2,ibUT2,ls2)
      statO  $\wedge$ 
      ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 
      same-xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
      pcOf cfg3 = startOfThenBranch  $\wedge$ 
      pcOf (last cfgs3) = elseBranch  $\wedge$ 
      misSpecL1 cfgs3
    ))

```

lemmas $\Delta 2$ -defs = $\Delta 2$ -def common-def PC-def same-xx-def inThenBranch-def
 elseBranch-def startOfThenBranch-def misSpecL1-def same-xx-def

lemma $\Delta 2$ -implies: $\Delta 2$ num (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \implies
 pcOf (last cfgs3) = 6 \wedge pcOf cfg3 = 4 \wedge
 pcOf (last cfgs4) = pcOf (last cfgs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 length cfgs3 = Suc 0 \wedge
 length cfgs3 = length cfgs4
 <proof>

definition $\Delta 3$:: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status
 \Rightarrow bool **where**

```

 $\Delta 3$  = ( $\lambda$  num
  (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
  (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)
  statO.
  (common (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 

```

$ls1 = ls3 \wedge ls2 = ls4 \wedge$
 $pcOf\ cfg3 = elseBranch \wedge$
 $pcOf\ (last\ cfigs3) = startOfThenBranch \wedge$
 $same\text{-}xx\ cfig3\ cfigs3\ cfig4\ cfigs4 \wedge$
 $misSpecL1\ cfigs3$
 $)$

lemmas $\Delta3\text{-}defs = \Delta3\text{-}def\ common\text{-}def\ PC\text{-}def\ same\text{-}xx\text{-}def\ elseBranch\text{-}def\ startOfThen\text{-}Branch\text{-}def$
 $misSpecL1\text{-}def\ same\text{-}xx\text{-}def$

lemma $\Delta3\text{-}implies: \Delta3\ num$

$(pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$
 $(cfig2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ (last\ cfigs3) = 4 \wedge pcOf\ cfig3 = 6 \wedge$
 $pcOf\ (last\ cfigs4) = pcOf\ (last\ cfigs3) \wedge$
 $pcOf\ cfig3 = pcOf\ cfig4 \wedge$
 $array\text{-}base\ aa1\ (getAvstore\ (stateOf\ (last\ cfigs3))) = array\text{-}base\ aa1\ (getAvstore$
 $(stateOf\ cfig3)) \wedge$
 $array\text{-}base\ aa1\ (getAvstore\ (stateOf\ (last\ cfigs4))) = array\text{-}base\ aa1\ (getAvstore$
 $(stateOf\ cfig4)) \wedge$
 $length\ cfigs3 = Suc\ 0 \wedge$
 $length\ cfigs3 = length\ cfigs4$
 $\langle proof \rangle$

definition $\Delta4 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta4 = (\lambda num$
 $(pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$
 $(cfig2, ibT2, ibUT2, ls2)$
 $statO.$
 $(pcOf\ cfig3 = endPC \wedge pcOf\ cfig4 = endPC \wedge cfigs3 = [] \wedge cfigs4 = [] \wedge$
 $pcOf\ cfig1 = endPC \wedge pcOf\ cfig2 = endPC))$

lemmas $\Delta4\text{-}defs = \Delta4\text{-}def\ common\text{-}def\ endPC\text{-}def$

lemma $init: initCond\ \Delta0$

$\langle proof \rangle$

lemma *step0*: *unwindIntoCond* Δ_0 (*oor* Δ_0 Δ_1)
<proof>

lemma *step1*: *unwindIntoCond* Δ_1 (*oor4* Δ_1 Δ_2 Δ_3 Δ_4)
<proof>

lemma *step2*: *unwindIntoCond* Δ_2 Δ_1
<proof>

lemma *step3*: *unwindIntoCond* Δ_3 (*oor* Δ_3 Δ_1)
<proof>

lemma *stepe*: *unwindIntoCond* Δ_4 Δ_4
<proof>

lemmas *theConds* = *step0 step1 step2 step3 stepe*

proposition *rsecure*
<proof>

end

10 Proof of Relative Security for fun3

theory *Fun3*
imports *../Instance-IMP/Instance-Secret-IMem*
Relative-Security.Unwinding-fin
begin

10.1 Function definition and Boilerplate

no-notation *bot* ($\langle \perp \rangle$)

consts *NN::nat*

lemma *NN::int* $NN \geq 0$ *<proof>*

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

consts *mispred* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool*

consts *update* :: *predState* \Rightarrow *pcounter list* \Rightarrow *predState*

consts *initPstate* :: *predState*

definition *aa1* :: *avname* **where** *aa1* = "a1"

definition *aa2* :: *avname* **where** *aa2* = "a2"

definition *vv* :: *avname* **where** *vv* = "v"

definition *xx* :: *avname* **where** *xx* = "x"

definition *tt* :: *avname* **where** *tt* = "t"

lemmas *vvars-defs* = *aa1-def aa2-def vv-def xx-def tt-def*

lemma *vvars-dff[simp]*:

aa1 ≠ *aa2* *aa1* ≠ *vv* *aa1* ≠ *xx* *aa1* ≠ *tt*

aa2 ≠ *aa1* *aa2* ≠ *vv* *aa2* ≠ *xx* *aa2* ≠ *tt*

vv ≠ *aa1* *vv* ≠ *aa2* *vv* ≠ *xx* *vv* ≠ *tt*

xx ≠ *aa1* *xx* ≠ *aa2* *xx* ≠ *vv* *xx* ≠ *tt*

tt ≠ *aa1* *tt* ≠ *aa2* *tt* ≠ *vv* *tt* ≠ *xx*

⟨*proof*⟩

fun *initAvstore* :: *avstore* ⇒ *bool* **where**

initAvstore (*Avstore as*) = (*as aa1* = (0, *size-aa1*) ∧ *as aa2* = (*size-aa1*, *size-aa2*))

fun *istate* :: *state* ⇒ *bool* **where**

istate *s* = (*initAvstore* (*getAvstore s*))

definition *prog* ≡

[

~~Start~~ ,

~~Input U xx~~ ,

~~tt ::= (N 0)~~ ,

~~IfJump (Less (V xx) (N NN)) 4 7~~ ,

~~vv ::= VA aa1 (V xx)~~ ,

~~Fence~~ ,

~~tt ::= (VA aa2 (Times (V vv) (N 512)))~~ ,

~~Output U (V tt)~~

]

lemma *cases-7*: (*i::pcounter*) = 0 ∨ *i* = 1 ∨ *i* = 2 ∨ *i* = 3 ∨ *i* = 4 ∨ *i* = 5 ∨
i = 6 ∨ *i* = 7 ∨ *i* > 7

⟨*proof*⟩

lemma *xx-NN-cases*: *vs xx* < *int NN* ∨ *vs xx* ≥ *int NN* ⟨*proof*⟩

lemma *is-If-pcOf[simp]*:

pcOf cfg < 8 ⇒ *is-IfJump* (*prog* ! (*pcOf cfg*)) ↔ *pcOf cfg* = 3

⟨*proof*⟩

lemma *is-If-pc*[simp]:
 $pc < 8 \implies is-IfJump (prog ! pc) \longleftrightarrow pc = 3$
 ⟨proof⟩

lemma *eq-Fence-pc*[simp]:
 $pc < 8 \implies prog ! pc = Fence \longleftrightarrow pc = 5$
 ⟨proof⟩

consts *resolve* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool*

interpretation *Prog-Mispred-Init* **where**
prog = *prog* **and** *initPstate* = *initPstate* **and**
mispred = *mispred* **and** *resolve* = *resolve* **and** *update* = *update* **and**
istate = *istate*
 ⟨proof⟩

abbreviation
 $stepB-abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
bool (**infix** $\langle \rightarrow B \rangle$ 55)
where $x \rightarrow B y == stepB\ x\ y$

abbreviation
 $stepsB-abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
bool (**infix** $\langle \rightarrow B^* \rangle$ 55)
where $x \rightarrow B^* y == star\ stepB\ x\ y$

abbreviation
 $stepM-abbrev :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
bool (**infix** $\langle \rightarrow M \rangle$ 55)
where $x \rightarrow M y == stepM\ x\ y$

abbreviation
 $stepN-abbrev :: config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist \times loc\ set \Rightarrow$
bool (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == stepN\ x\ y$

abbreviation
 $stepsN-abbrev :: config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist \times loc\ set \Rightarrow$
bool (**infix** $\langle \rightarrow N^* \rangle$ 55)
where $x \rightarrow N^* y == star\ stepN\ x\ y$

abbreviation

stepS-abbrev :: *configS* \Rightarrow *configS* \Rightarrow *bool* (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == \text{stepS } x y$

abbreviation

stepsS-abbrev :: *configS* \Rightarrow *configS* \Rightarrow *bool* (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == \text{star stepS } x y$

lemma *endPC[simp]*: *endPC* = 8
 $\langle \text{proof} \rangle$

lemma *is-getTrustedInput-pcOf[simp]*: *pcOf cfg* < 8 \implies *is-getInput* (*prog!*(*pcOf* *cfg*)) \longleftrightarrow *pcOf* *cfg* = 1
 $\langle \text{proof} \rangle$

lemma *getUntrustedInput-pcOf[simp]*: *prog!1* = *Input U xx*
 $\langle \text{proof} \rangle$

lemma *getInput-not3[simp]*: $\neg \text{is-getInput } (\text{prog } ! 3)$
 $\langle \text{proof} \rangle$

lemma *getInput-not4[simp]*: $\neg \text{is-getInput } (\text{prog } ! 4)$
 $\langle \text{proof} \rangle$

lemma *Output-not4[simp]*: $\neg \text{is-Output } (\text{prog } ! 4)$
 $\langle \text{proof} \rangle$

lemma *is-Output-pcOf[simp]*: *pcOf cfg* < 8 \implies *is-Output* (*prog!*(*pcOf* *cfg*)) \longleftrightarrow *pcOf* *cfg* = 7
 $\langle \text{proof} \rangle$

lemma *is-Output*: *is-Output* (*prog* ! 7)
 $\langle \text{proof} \rangle$

lemma *is-Fence[simp]*: (*prog* ! 5) = *Fence*
 $\langle \text{proof} \rangle$

lemma *not-is-getTrustedInput[simp]*: *cfg* = *Config 3* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*) \implies $\neg \text{is-getInput } (\text{prog } ! \text{pcOf } \text{cfg})$
 $\langle \text{proof} \rangle$

lemma *not-is-Output[simp]*: *cfg* = *Config pc* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*) \implies
 $\text{pc} = 3 \implies \neg \text{is-Output } (\text{prog } ! \text{pcOf } \text{cfg})$

$\langle \text{proof} \rangle$

lemma *isSecV-pcOf[simp]*:
 $\text{isSecV } (cfg, ibT, ibUT) \longleftrightarrow \text{pcOf } cfg = 0$
 $\langle \text{proof} \rangle$

lemma *isSecO-pcOf[simp]*:
 $\text{isSecO } (pstate, cfg, cfs, ibT, ibUT, ls) \longleftrightarrow (\text{pcOf } cfg = 0 \wedge \text{cfs} = [])$
 $\langle \text{proof} \rangle$

lemma *getInputT-not[simp]*: $\text{pcOf } cfg < 8 \implies$
 $(\text{prog} ! \text{pcOf } cfg) \neq \text{Input } T \text{ inp}$
 $\langle \text{proof} \rangle$

lemma *getActV-pcOf[simp]*:
 $\text{pcOf } cfg < 8 \implies$
 $\text{getActV } (cfg, ibT, ibUT, ls) =$
 $(\text{if } \text{pcOf } cfg = 1 \text{ then } \text{lhs } ibUT \text{ else } \perp)$
 $\langle \text{proof} \rangle$

lemma *getObsV-pcOf[simp]*:
 $\text{pcOf } cfg < 8 \implies$
 $\text{getObsV } (cfg, ibT, ibUT, ls) =$
 $(\text{if } \text{pcOf } cfg = 7 \text{ then}$
 $\quad (\text{outOf } (\text{prog}!(\text{pcOf } cfg)) (\text{stateOf } cfg), ls)$
 $\quad \text{else } \perp$
 $\quad)$
 $\langle \text{proof} \rangle$

lemma *getActO-pcOf[simp]*:
 $\text{pcOf } cfg < 8 \implies$
 $\text{getActO } (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(\text{if } \text{pcOf } cfg = 1 \wedge \text{cfs} = [] \text{ then } \text{lhs } ibUT \text{ else } \perp)$
 $\langle \text{proof} \rangle$

lemma *getObsO-pcOf[simp]*:
 $\text{pcOf } cfg < 8 \implies$
 $\text{getObsO } (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(\text{if } (\text{pcOf } cfg = 7 \wedge \text{cfs} = []) \text{ then}$
 $\quad (\text{outOf } (\text{prog}!(\text{pcOf } cfg)) (\text{stateOf } cfg), ls)$
 $\quad \text{else } \perp$
 $\quad)$
 $\langle \text{proof} \rangle$

lemma *eqSec-pcOf[simp]*:
 $eqSec\ (cfg1, ibT, ibUT1, ls1)\ (pstate3, cfg3, cfigs3, ibT, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfigs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
 $\langle proof \rangle$

lemma *nextB-pc0[simp]*:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc0[simp]*:
 $readLocs\ (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc1[simp]*:
 $ibUT \neq LNil \implies nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p, ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc1[simp]*:
 $readLocs\ (Config\ 1\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc1'[simp]*:
 $ibUT \neq LNil \implies nextB\ (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
 $=$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p, ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc1'[simp]*:
 $readLocs\ (Config\ (Suc\ 0)\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc2[simp]*:
 $nextB\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 3\ (State\ (Vstore\ (vs(tt := 0))))\ avst\ h\ p, ibT, ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc2[simp]*:
 $readLocs\ (Config\ 2\ s) = \{\}$

$\langle \text{proof} \rangle$

lemma *nextB-pc3-then[simp]*:

$vs\ xx < int\ NN \implies$

$nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *nextB-pc3-else[simp]*:

$vs\ xx \geq int\ NN \implies$

$nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 7\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *nextB-pc3*:

$nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$

$(Config\ (if\ vs\ xx < NN\ then\ 4\ else\ 7)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *nextM-pc3-then[simp]*:

$vs\ xx \geq int\ NN \implies$

$nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *nextM-pc3-else[simp]*:

$vs\ xx < int\ NN \implies$

$nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 7\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *nextM-pc3*:

$nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$

$(Config\ (if\ vs\ xx < NN\ then\ 7\ else\ 4)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$

$\langle \text{proof} \rangle$

lemma *readLocs-pc3[simp]*:

$readLocs\ (Config\ 3\ s) = \{\}$

$\langle \text{proof} \rangle$

lemma *nextB-pc4[simp]*:

$nextB\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT) =$

$(let\ l = array-loc\ aa1\ (nat\ (vs\ xx))\ avst$

$in\ (Config\ 5\ (State\ (Vstore\ (vs\ (v := h\ l)))\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT)$

$\langle proof \rangle$

lemma *readLocs-pc4*[simp]:

$readLocs (Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa1\ (nat\ (vs\ xx))\ avst\}$

$\langle proof \rangle$

lemma *nextB-pc5*[simp]:

$nextB (Config\ 5\ s,\ ibT,\ ibUT) = (Config\ 6\ s,\ ibT,\ ibUT)$

$\langle proof \rangle$

lemma *readLocs-pc5*[simp]:

$readLocs (Config\ 5\ s) = \{\}$

$\langle proof \rangle$

lemma *nextB-pc6*[simp]:

$nextB (Config\ 6\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT) =$

$(let\ l = array-loc\ aa2\ (nat\ (vs\ vv * 512))\ avst$

$in\ (Config\ 7\ (State\ (Vstore\ (vs(tt := h\ l)))\ avst\ (Heap\ h)\ p)),\ ibT,\ ibUT)$

$\langle proof \rangle$

lemma *readLocs-pc6*[simp]:

$readLocs (Config\ 6\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa2\ (nat\ (vs\ vv * 512))\ avst\}$

$\langle proof \rangle$

lemma *nextB-pc7*[simp]:

$nextB (Config\ 7\ s,\ ibT,\ ibUT) = (Config\ 8\ s,\ ibT,\ ibUT)$

$\langle proof \rangle$

lemma *readLocs-pc7*[simp]:

$readLocs (Config\ 7\ s) = \{\}$

$\langle proof \rangle$

lemma *nextB-stepB-pc*:

$pc < 8 \implies (pc = 1 \implies ibUT \neq LNil) \implies$

$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ nextB (Config\ pc\ s,\ ibT,\ ibUT)$

$\langle proof \rangle$

lemma *not-finalB*:

$pc < 8 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
 <proof>

lemma *finalB-pc-iff'*:

$pc < 8 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil)$
 <proof>

lemma *finalB-pc-iff*:

$pc \leq 8 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil \vee pc = 8)$
 <proof>

lemma *finalB-pcOf-iff[simp]*:

$pcOf\ cfg \leq 8 \implies$
 $finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = 8)$
 <proof>

lemma *finalS-cond:pcOf cfg < 8 \implies cfgs = [] \implies (pcOf cfg = 1 \longrightarrow ibUT \neq LNil) \implies \neg finalS (pstate, cfg, cfgs, ibT, ibUT, ls)*
 <proof>

lemma *finalS-cond-spec*:

$pcOf\ cfg < 8 \implies$
 $((pcOf (last\ cfgs) = 4 \vee pcOf (last\ cfgs) = 5) \wedge pcOf\ cfg = 7) \vee$
 $(pcOf (last\ cfgs) = 7 \wedge pcOf\ cfg = 4) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg finalS (pstate, cfg, cfgs, ibT, ibUT, ls)$
 <proof>

end

10.2 Proof

theory *Fun3-secure*

imports *Fun3*

begin

type-synonym *stateO* = *configS*

type-synonym *stateV* = *config* \times *val llist* \times *val llist* \times *loc set*

definition *PC* \equiv {0..7}

definition $beforeInput = \{0,1\}$
definition $afterInput = \{2,3,4,5,6,7\}$
definition $startOfThenBranch = 4$
definition $inThenBranchBeforeFence = \{4,5\}$
definition $elseBranch = 7$
definition $beforeFence = \{2..4\}$
definition $beforeAssign-vv = \{0..4\}$

definition $common :: stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$

where

$common = (\lambda$
 $(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(pstate3 = pstate4 \wedge$
 $cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge$
 $pcOf\ cfg3 = pcOf\ cfg4 \wedge map\ pcOf\ cfgs3 = map\ pcOf\ cfgs4 \wedge$
 $pcOf\ cfg3 \in PC \wedge pcOf\ ' (set\ cfgs3) \subseteq PC \wedge$
 $///$
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg3)) = array-base\ aa1\ (getAvstore\ (stateOf\ cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfgs3. array-base\ aa1\ (getAvstore\ (stateOf\ cfg3')) = array-base\ aa1\ (getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfgs4. array-base\ aa1\ (getAvstore\ (stateOf\ cfg4')) = array-base\ aa1\ (getAvstore\ (stateOf\ cfg4))) \wedge$
 $array-base\ aa2\ (getAvstore\ (stateOf\ cfg3)) = array-base\ aa2\ (getAvstore\ (stateOf\ cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfgs3. array-base\ aa2\ (getAvstore\ (stateOf\ cfg3')) = array-base\ aa2\ (getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfgs4. array-base\ aa2\ (getAvstore\ (stateOf\ cfg4')) = array-base\ aa2\ (getAvstore\ (stateOf\ cfg4))) \wedge$
 $///$
 $(statA = Diff \longrightarrow statO = Diff)))$

lemma $common-implies: common$

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \Longrightarrow$

$pcOf\ cfg1 < 9 \wedge pcOf\ cfg2 = pcOf\ cfg1$
 ⟨proof⟩

definition $\Delta 0 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 0 = (\lambda num$
 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (common (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge
 ibUT1 = ibUT3 \wedge ibUT2 = ibUT4 \wedge
 (pcOf cfg3 > 1 \longrightarrow same-var-o xx cfg3 cfs3 cfg4 cfs4) \wedge
 (pcOf cfg3 < 2 \longrightarrow ibUT1 \neq LNil \wedge ibUT2 \neq LNil \wedge ibUT3 \neq LNil \wedge ibUT4 \neq LNil)
 \wedge
 pcOf cfg3 \in beforeInput \wedge
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 noMisSpec cfs3
))

lemmas $\Delta 0\text{-defs} = \Delta 0\text{-def common-def PC-def beforeInput-def noMisSpec-def same-var-o-def}$

lemma $\Delta 0\text{-implies: } \Delta 0\ num$

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 (pcOf cfg3 = 1 \longrightarrow ibUT3 \neq LNil) \wedge
 (pcOf cfg4 = 1 \longrightarrow ibUT4 \neq LNil) \wedge
 pcOf cfg1 < 8 \wedge pcOf cfg2 = pcOf cfg1 \wedge
 cfs3 = [] \wedge pcOf cfg3 < 8 \wedge
 cfs4 = [] \wedge pcOf cfg4 < 8
 ⟨proof⟩

definition $\Delta 1 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 1 = (\lambda num$

```

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO.
(common
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO  $\wedge$ 
pcOf cfg3  $\in$  afterInput  $\wedge$ 
same-var-o xx cfg3 cfs3 cfg4 cfs4  $\wedge$ 
ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 
noMisSpec cfs3
))

```

lemmas $\Delta 1$ -defs = $\Delta 1$ -def common-def PC-def afterInput-def same-var-o-def noMisSpec-def

lemma $\Delta 1$ -implies: $\Delta 1$ num

```

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO  $\implies$ 
pcOf cfg1 < 8  $\wedge$ 
cfs3 = []  $\wedge$  pcOf cfg3  $\neq$  1  $\wedge$  pcOf cfg3 < 8  $\wedge$ 
cfs4 = []  $\wedge$  pcOf cfg4  $\neq$  1  $\wedge$  pcOf cfg4 < 8
<proof>

```

definition $\Delta 2$:: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status
 \Rightarrow bool **where**

```

 $\Delta 2$  = ( $\lambda$ num
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO.
(common
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA

```

(cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \wedge
 pcOf cfg3 = startOfThenBranch \wedge
 pcOf (last cfs3) = elseBranch \wedge
 same-var-o xx cfg3 cfs3 cfg4 cfs4 \wedge
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 misSpecL1 cfs3
))

lemmas $\Delta 2$ -defs = $\Delta 2$ -def common-def PC-def same-var-def startOfThenBranch-def

misSpecL1-def elseBranch-def

lemma $\Delta 2$ -implies: $\Delta 2$ num

(pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \implies
 pcOf (last cfs3) = 7 \wedge pcOf cfg3 = 4 \wedge
 pcOf (last cfs4) = pcOf (last cfs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 length cfs3 = Suc 0 \wedge
 length cfs3 = length cfs4
 <proof>

definition $\Delta 3$:: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status
 \Rightarrow bool **where**

$\Delta 3$ = (λ num
 (pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO.
 (common (pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \wedge
 pcOf cfg3 = elseBranch \wedge
 pcOf (last cfs3) \in inThenBranchBeforeFence \wedge
 same-var-o xx cfg3 cfs3 cfg4 cfs4 \wedge
 Language-Prelims.dist ls3 ls4 \subseteq Language-Prelims.dist ls1 ls2 \wedge
 (pcOf (last cfs3) = 4 \longrightarrow ls1 = ls3 \wedge ls2 = ls4) \wedge

misSpecL1 cfgs3
))

lemmas $\Delta 3$ -defs = $\Delta 3$ -def common-def PC-def inThenBranchBeforeFence-def
 beforeAssign-vv-def misSpecL1-def elseBranch-def
 same-var-o-def

lemma $\Delta 3$ -implies: $\Delta 3$ num

(*pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3*)
 (*pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4*)
 statA
 (*cfg1, ibT1, ibUT1, ls1*)
 (*cfg2, ibT2, ibUT2, ls2*)
 statO \implies
 (*pcOf (last cfgs3) = 4 \vee pcOf (last cfgs3) = 5*) \wedge *pcOf cfg3 = 7* \wedge
pcOf (last cfgs4) = pcOf (last cfgs3) \wedge
pcOf cfg3 = pcOf cfg4 \wedge
array-base aa1 (getAvstore (stateOf (last cfgs3))) = array-base aa1 (getAvstore
(stateOf cfg3)) \wedge
array-base aa1 (getAvstore (stateOf (last cfgs4))) = array-base aa1 (getAvstore
(stateOf cfg4)) \wedge
length cfgs3 = Suc 0 \wedge
length cfgs3 = length cfgs4 \wedge
vstore (getVstore (stateOf (last cfgs3))) xx = vstore (getVstore (stateOf (last
cfgs4))) xx
 <proof>

definition $\Delta 1'$:: *enat* \Rightarrow *stateO* \Rightarrow *stateO* \Rightarrow *status* \Rightarrow *stateV* \Rightarrow *stateV* \Rightarrow *status*
 \Rightarrow *bool* **where**

$\Delta 1' = (\lambda \text{num}$
 (*pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3*)
 (*pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4*)
 statA
 (*cfg1, ibT1, ibUT1, ls1*)
 (*cfg2, ibT2, ibUT2, ls2*)
 statO.
 (common
 (*pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3*)
 (*pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4*)
 statA
 (*cfg1, ibT1, ibUT1, ls1*)
 (*cfg2, ibT2, ibUT2, ls2*)
 statO \wedge
pcOf cfg3 = elseBranch \wedge
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 \wedge
Language-Prelims.dist ls3 ls4 \subseteq Language-Prelims.dist ls1 ls2 \wedge

noMisSpec cfgs3
))

lemmas $\Delta 1'$ -defs = $\Delta 1'$ -def common-def PC-def afterInput-def same-var-o-def
noMisSpec-def
elseBranch-def

lemma $\Delta 1'$ -implies: $\Delta 1'$ num
 (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 pcOf cfg1 < 8 \wedge
 cfgs3 = [] \wedge pcOf cfg3 \neq 1 \wedge pcOf cfg3 < 8 \wedge
 cfgs4 = [] \wedge pcOf cfg4 \neq 1 \wedge pcOf cfg4 < 8
 <proof>

definition $\Delta 4$:: enat \implies stateO \implies stateO \implies status \implies stateV \implies stateV \implies status
 \implies bool **where**

$\Delta 4$ = (λ num
 (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (pcOf cfg3 = endPC \wedge pcOf cfg4 = endPC \wedge cfgs3 = [] \wedge cfgs4 = [] \wedge
 pcOf cfg1 = endPC \wedge pcOf cfg2 = endPC))

lemmas $\Delta 4$ -defs = $\Delta 4$ -def common-def endPC-def

lemma *init*: *initCond* $\Delta 0$
 <proof>

lemma *step0*: *unwindIntoCond* $\Delta 0$ (*oor* $\Delta 0$ $\Delta 1$)
 <proof>

lemma *step1*: *unwindIntoCond* $\Delta 1$ (*oor4* $\Delta 1$ $\Delta 2$ $\Delta 3$ $\Delta 4$)
 <proof>

lemma *step2*: *unwindIntoCond* Δ_2 Δ_1
<proof>

lemma *step3*: *unwindIntoCond* Δ_3 (*oor* Δ_3 Δ_1')
<proof>

lemma *step1'*: *unwindIntoCond* Δ_1' Δ_4
<proof>

lemma *step0*: *unwindIntoCond* Δ_4 Δ_4
<proof>

lemmas *theConds* = *step0 step1 step2 step3 step1' step0*

proposition *rsecure*
<proof>
end

11 Proof of Relative Security for fun4

theory *Fun4*
imports *../Instance-IMP/Instance-Secret-IMem*
Relative-Security.Unwinding-fin
begin

11.1 Function definition and Boilerplate

no-notation *bot* ($\langle \perp \rangle$)

consts *NN* :: *nat*
consts *size-aa1* :: *nat*
consts *size-aa2* :: *nat*
lemma *NN*: *int* *NN* ≥ 0 *<proof>*

locale *array-nempty* = **assumes** *aa1*:*size-aa1* > 0 **and** *NN*: *int* *NN* > 0

definition *aa1* :: *avname* **where** *aa1* = "*a1*"
definition *aa2* :: *avname* **where** *aa2* = "*a2*"
definition *vv* :: *avname* **where** *vv* = "*v*"

definition $xx :: \text{avname}$ **where** $xx = 'i'$
definition $tt :: \text{avname}$ **where** $tt = 'w'$

lemmas $vvars-defs = aa1-def\ aa2-def\ vv-def\ xx-def\ tt-def$

lemma $vvars-dff[simp]$:

$aa1 \neq aa2\ aa1 \neq vv\ aa1 \neq xx\ aa1 \neq tt$
 $aa2 \neq aa1\ aa2 \neq vv\ aa2 \neq xx\ aa2 \neq tt$
 $vv \neq aa1\ vv \neq aa2\ vv \neq xx\ vv \neq tt$
 $xx \neq aa1\ xx \neq aa2\ xx \neq vv\ xx \neq tt$
 $tt \neq aa1\ tt \neq aa2\ tt \neq vv\ tt \neq xx$
 $\langle proof \rangle$

fun $initAvstore :: \text{avstore} \Rightarrow \text{bool}$ **where**

$initAvstore\ (Avstore\ as) = (as\ aa1 = (0, size-aa1) \wedge as\ aa2 = (size-aa1, size-aa2))$

fun $istate :: \text{state} \Rightarrow \text{bool}$ **where**

$istate\ s = (initAvstore\ (getAvstore\ s))$

definition $prog \equiv$

[
 \emptyset Start ,
 $\not\emptyset$ Input $U\ xx$,
 $\not\emptyset$ $tt ::= (N\ 0)$,
 $\not\emptyset$ IfJump (Less (V xx) (N NN)) 4 6 ,
 $\not\emptyset$ $vv ::= VA\ aa1\ (N\ 0)$,
 $\not\emptyset$ $tt ::= Plus\ (VA\ aa2\ (Times\ (V\ vv)\ (N\ 512)))\ (V\ xx)$,
 $\not\emptyset$ Output $U\ (V\ tt)$
]

lemma $cases-6: (i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee i = 6 \vee i > 6$
 $\langle proof \rangle$

lemma $cases-thenBranch: (i::pcounter) < 4 \vee i = 4 \vee i = 5 \vee i = 6 \vee i > 6$
 $\langle proof \rangle$

lemma $xx-NN-cases: vs\ xx < int\ NN \vee vs\ xx \geq int\ NN\ \langle proof \rangle$

lemma $is-If-pcOf[simp]$:

$pcOf\ cfg < 7 \implies is-IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 3$

<proof>

lemma *is-If-pc[simp]*:

$pc < 7 \implies is-IfJump (prog ! pc) \longleftrightarrow pc = 3$

<proof>

lemma *is-If-pcThen[simp]*: $pcOf\ cf\ g \in \{4..6\} \implies \neg is-IfJump (prog ! pcOf\ cf\ g)$

<proof>

consts *mispred* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$
consts *resolve* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$
consts *update* :: $predState \Rightarrow pcounter\ list \Rightarrow predState$
consts *initPstate* :: $predState$

interpretation *Prog-Mispred-Init* **where**

prog = *prog* **and** *initPstate* = *initPstate* **and**

mispred = *mispred* **and** *resolve* = *resolve* **and** *update* = *update* **and**

istate = *istate*

<proof>

abbreviation

stepB-abbrev :: $config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow bool$ (**infix** $\langle \rightarrow B \rangle$ 55)

where $x \rightarrow B\ y == stepB\ x\ y$

abbreviation

stepsB-abbrev :: $config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow bool$ (**infix** $\langle \rightarrow B^* \rangle$ 55)

where $x \rightarrow B^*\ y == star\ stepB\ x\ y$

abbreviation

stepM-abbrev :: $config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow bool$ (**infix** $\langle \rightarrow M \rangle$ 55)

where $x \rightarrow M\ y == stepM\ x\ y$

abbreviation

stepN-abbrev :: $config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist \times loc\ set \Rightarrow bool$ (**infix** $\langle \rightarrow N \rangle$ 55)

where $x \rightarrow N\ y == stepN\ x\ y$

abbreviation

stepsN-abbrev :: $config \times val\ llist \times val\ llist \times loc\ set \Rightarrow config \times val\ llist \times val\ llist \times loc\ set \Rightarrow bool$ (**infix** $\langle \rightarrow N^* \rangle$ 55)

where $x \rightarrow N^* y == \text{star } \text{stepN } x y$

abbreviation

$\text{stepS-abbrev} :: \text{configS} \Rightarrow \text{configS} \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == \text{stepS } x y$

abbreviation

$\text{stepsS-abbrev} :: \text{configS} \Rightarrow \text{configS} \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == \text{star } \text{stepS } x y$

lemma $\text{endPC}[simp]: \text{endPC} = 7$
<proof>

lemma $\text{is-getUntrustedInput-pcOf}[simp]: \text{pcOf } \text{cfg} < 7 \implies \text{is-getInput } (\text{prog}!(\text{pcOf } \text{cfg})) \longleftrightarrow \text{pcOf } \text{cfg} = 1$
<proof>

lemma $\text{getUntrustedInput-pcOf}[simp]: \text{prog}!1 = \text{Input } U \text{ } xx$
<proof>

lemma $\text{is-getTrustedInput}[simp]: \text{is-getInput } (\text{prog} ! 1)$
<proof>

lemma $\text{getInput-not4}[simp]: \neg \text{is-getInput } (\text{prog} ! 4)$
<proof>

lemma $\text{getInput-not5}[simp]: \neg \text{is-getInput } (\text{prog} ! 5)$
<proof>

lemma $\text{OutputT-not6}[simp]: (\text{prog} ! 6) = \text{Output } U (V \text{ } tt)$
<proof>

lemma $\text{is-Output-pcOf}[simp]: \text{pcOf } \text{cfg} < 7 \implies \text{is-Output } (\text{prog}!(\text{pcOf } \text{cfg})) \longleftrightarrow \text{pcOf } \text{cfg} = 6$
<proof>

lemma $\text{is-Fence-pcOf}[simp]: \text{pcOf } \text{cfg} < 7 \implies \text{prog} ! (\text{pcOf } \text{cfg}) \neq \text{Fence}$
<proof>

lemma $\text{is-Fence-pcThen}[simp]: 3 \leq \text{pcOf } \text{cfg} \wedge \text{pcOf } \text{cfg} \leq 5 \implies (\text{prog} ! \text{pcOf } \text{cfg}) \neq \text{Fence}$
<proof>

lemma $\text{is-Output}[simp]: \text{is-Output } (\text{prog} ! 6)$
<proof>

lemma *getInput-not*[intro]:*is-getInput* (prog ! 4) \implies False <proof>

lemma *Output-not4*[intro]:*is-Output* (prog ! 4) \implies False <proof>

lemma *Fence-not4*[intro]:prog ! 4 = Fence \implies False <proof>

lemma *getInput-not55*[intro]:*is-getInput* (prog ! 5) \implies False <proof>

lemma *Output-not5*[intro]:*is-Output* (prog ! 5) \implies False <proof>

lemma *Fence-not5*[intro]:prog ! 5 = Fence \implies False <proof>

lemma *Jump-not6*: \neg *is-IfJump* (prog ! 6)<proof>

lemma *isSecV-pcOf*[simp]:

isSecV (cfg,ibT,ibUT) \longleftrightarrow pcOf cfg = 0
<proof>

lemma *isSecO-pcOf*[simp]:

isSecO (pstate,cfg,cfgs,ibT,ibUT,ls) \longleftrightarrow (pcOf cfg = 0 \wedge cfgs = [])
<proof>

lemma *inputT-not*[simp]: pcOf cfg < 7 \implies

(prog ! pcOf cfg) \neq Input T inp

<proof>

lemma *getActV-pcOf*[simp]:

pcOf cfg < 7 \implies
getActV (cfg,ibT,ibUT,ls) =
(if pcOf cfg = 1 then lhd ibUT else \perp)
<proof>

lemma *getObsV-pcOf*[simp]:

pcOf cfg < 7 \implies
getObsV (cfg,ibT,ibUT,ls) =
(if pcOf cfg = 6 then
(outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
else \perp
)
<proof>

lemma *getObsV-pcOf6*[simp]:

pcOf cfg = 6 \implies
getObsV (cfg,ibT,ibUT,ls) =
(outOf (prog!(pcOf cfg)) (stateOf cfg), ls)

<proof>

lemma *getActO-pcOf*[simp]:

pcOf cfg < 7 \implies

$getActO (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1 \wedge cfs = []\ then\ lhd\ ibUT\ else\ \perp)$
 $\langle proof \rangle$

lemma $getObsO-pcOf[simp]$:
 $pcOf\ cfg < 7 \implies$
 $getObsO (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 6 \wedge cfs = [])\ then$
 $(outOf (prog!(pcOf\ cfg)) (stateOf\ cfg), ls)$
 $else\ \perp$
 $)$
 $\langle proof \rangle$

lemma $eqSec-pcOf[simp]$:
 $eqSec (cfg1, ibT, ibUT1, ls1) (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
 $\langle proof \rangle$

lemma $nextB-pc0[simp]$:
 $nextB (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc0[simp]$:
 $readLocs (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc1[simp]$:
 $ibUT \neq LNil \implies nextB (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p, ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc1[simp]$:
 $readLocs (Config\ 1\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc1'[simp]$:
 $ibUT \neq LNil \implies nextB (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
 $=$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p, ibT, ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc1* [simp]:
 $readLocs (Config (Suc 0) s) = \{\}$
 ⟨proof⟩

lemma *nextB-pc2* [simp]:
 $nextB (Config 2 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config 3 (State (Vstore (vs(tt := 0))) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *readLocs-pc2* [simp]:
 $readLocs (Config 2 s) = \{\}$
 ⟨proof⟩

lemma *nextB-pc3-then* [simp]:
 $vs\ xx < int\ NN \implies$
 $nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config 4 (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *nextB-pc3-else* [simp]:
 $vs\ xx \geq int\ NN \implies$
 $nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config 6 (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *nextB-pc3*:
 $nextB (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config (if\ vs\ xx < int\ NN\ then\ 4\ else\ 6) (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *nextM-pc3-then* [simp]:
 $vs\ xx \geq int\ NN \implies$
 $nextM (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config 4 (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *nextM-pc3-else* [simp]:
 $vs\ xx < int\ NN \implies$
 $nextM (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config 6 (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *nextM-pc3*:
 $nextM (Config 3 (State (Vstore vs) avst h p), ibT, ibUT) =$
 $(Config (if\ vs\ xx < int\ NN\ then\ 6\ else\ 4) (State (Vstore vs) avst h p), ibT, ibUT)$
 ⟨proof⟩

lemma *readLocs-pc3*[simp]:
 $readLocs (Config\ 3\ s) = \{\}$
 ⟨proof⟩

lemma *nextB-pc4*[simp]:
 $nextB (Config\ 4 (State (Vstore\ vs)\ avst (Heap\ h)\ p),\ ibT,ibUT) =$
 $(let\ l = array-loc\ aa1\ 0\ avst$
 $in (Config\ 5 (State (Vstore (vs(vv := h\ l)))\ avst (Heap\ h)\ p)),\ ibT,ibUT)$
 ⟨proof⟩

lemma *readLocs-pc4*[simp]:
 $readLocs (Config\ 4 (State (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa1\ 0\ avst\}$
 ⟨proof⟩

lemma *nextB-pc5*[simp]:
 $nextB (Config\ 5 (State (Vstore\ vs)\ avst (Heap\ h)\ p),\ ibT,ibUT) =$
 $(let\ l = array-loc\ aa2 (nat (vs\ vv * 512))\ avst$
 $in (Config\ 6 (State (Vstore (vs(tt := h\ l + vs\ xx)))\ avst (Heap\ h)\ p)),\ ibT,ibUT)$
 ⟨proof⟩

lemma *readLocs-pc5*[simp]:
 $readLocs (Config\ 5 (State (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa2 (nat (vs\ vv * 512))\ avst\}$
 ⟨proof⟩

lemma *nextB-pc6*[simp]:
 $nextB (Config\ 6\ s,\ ibT,ibUT) = (Config\ 7\ s,\ ibT,ibUT)$
 ⟨proof⟩

lemma *readLocs-pc6*[simp]:
 $readLocs (Config\ 6 (State (Vstore\ vs)\ avst\ h\ p)) = \{\}$
 ⟨proof⟩

lemma *nextB-stepB-pc*:
 $pc < 7 \implies (pc = 1 \implies ibUT \neq LNil) \implies$
 $(Config\ pc\ s,\ ibT,ibUT) \rightarrow B\ nextB (Config\ pc\ s,\ ibT,ibUT)$
 ⟨proof⟩

lemma *nextB-avst-consistent-aux*:
 $4 \leq pc \wedge pc \leq 6 \implies$

$(nextB (Config\ pc\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p), ibT, ibUT)) = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ (Heap\ h')\ p'), ibT, ibUT') \implies$
 $avst = avst' \wedge$
 $vs\ xx = vs'\ xx \wedge$
 $h = h'$
 <proof>

lemma *nextB-avst-consistent*:

$4 \leq pcOf\ cfg \wedge pcOf\ cfg \leq 6 \implies$
 $(nextB (cfg, ibT, ibUT)) = (cfg', ibT, ibUT') \implies$
 $(getAvstore (stateOf\ cfg)) = (getAvstore (stateOf\ cfg')) \wedge$
 $(getHheap (stateOf\ cfg)) = (getHheap (stateOf\ cfg')) \wedge$
 $vstore (getVstore (stateOf\ cfg))\ xx = vstore (getVstore (stateOf\ cfg'))\ xx$
 <proof>

lemma *nextB-pcs-consistent*:

$4 \leq pcOf\ cfg1 \wedge pcOf\ cfg1 \leq 6 \implies pcOf\ cfg1 = pcOf\ cfg2 \implies$
 $(nextB (cfg1, ibT1, ibUT1)) = (cfg1', ibT1', ibUT1') \implies$
 $(nextB (cfg2, ibT2, ibUT2)) = (cfg2', ibT2', ibUT2') \implies$
 $pcOf\ cfg1' = pcOf\ cfg2'$
 <proof>

lemma *not-finalB*:

$pc < 7 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
 <proof>

lemma *finalB-pc-iff'*:

$pc < 7 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \iff$
 $(pc = 1 \wedge ibUT = LNil)$
 <proof>

lemma *finalB-pc-iff*:

$pc \leq 7 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \iff$
 $(pc = 1 \wedge ibUT = LNil \vee pc = 7)$
 <proof>

lemma *finalB-pcOf-iff[simp]*:

$pcOf\ cfg \leq 7 \implies$
 $finalB (cfg, ibT, ibUT) \iff (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = 7)$
 <proof>

lemma *finalS-cond:pcOf\ cfg < 7 \implies cfs = [] \implies (pcOf\ cfg = 1 \longrightarrow ibUT \neq*

$LNil \implies \neg \text{finalS } (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *finalS-cond-spec*:

$pcOf\ cfg < 7 \implies$
 $((pcOf\ (last\ cfgs) = 4 \vee pcOf\ (last\ cfgs) = 5 \vee pcOf\ (last\ cfgs) = 6) \wedge pcOf\$
 $cfg = 6) \vee (pcOf\ (last\ cfgs) = 6 \wedge pcOf\ cfg = 4) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg \text{finalS } (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

end

11.2 Proof

theory *Fun4-secure*
imports *Fun4*
begin

definition $PC \equiv \{0..6\}$

definition *same-xx-cp* $cfg1\ cfg2 \equiv$

$vstore\ (getVstore\ (stateOf\ cfg1))\ xx = vstore\ (getVstore\ (stateOf\ cfg2))\ xx$
 $\wedge\ vstore\ (getVstore\ (stateOf\ cfg1))\ xx = 0$

definition *common-memory* $cfg\ cfg'\ cfgs' \equiv$

$array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg)) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\$
 $cfg')) \wedge$
 $(\forall\ cfg'' \in set\ cfgs'.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg'')) = array\text{-}base\ aa1$
 $(getAvstore\ (stateOf\ cfg))) \wedge$
 $array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg)) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\$
 $cfg')) \wedge$
 $(\forall\ cfg'' \in set\ cfgs'.\ array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg'')) = array\text{-}base\ aa2$
 $(getAvstore\ (stateOf\ cfg))) \wedge$
 $(getHheap\ (stateOf\ cfg)) = (getHheap\ (stateOf\ cfg')) \wedge$
 $(\forall\ cfg'' \in set\ cfgs'.\ getHheap\ (stateOf\ cfg) = (getHheap\ (stateOf\ cfg''))) \wedge$
 $(getAvstore\ (stateOf\ cfg)) = (getAvstore\ (stateOf\ cfg'))$

definition *beforeInput* $= \{0,1\}$

definition *afterInput* $= \{2..6\}$

definition *elseBranch* $= 6$

definition *startOfThenBranch* $= 4$

definition *inThenBranch* $= \{4..6\}$

definition *afterInputNotInElse* = {2,3,4,5,6,8}

definition *inThenBranchBeforeOutput* = {3,4,5}

definition *atCond* = 3

definition *atThenOutput* = 5

definition *atJump* = 6

definition *common-strat1* :: *stateO* \Rightarrow *stateO* \Rightarrow *status* \Rightarrow *stateV* \Rightarrow *stateV* \Rightarrow *status* \Rightarrow *bool*

where

common-strat1 =

(λ (*pstate3*, *cfg3*, *cfgs3*, *ibT3*, *ibUT3*, *ls3*)

 (*pstate4*, *cfg4*, *cfgs4*, *ibT4*, *ibUT4*, *ls4*)

statA

 (*cfg1*, *ibT1*, *ibUT1*, *ls1*)

 (*cfg2*, *ibT2*, *ibUT2*, *ls2*)

statO.)

(*pstate3* = *pstate4* \wedge

cfg1 = *cfg3* \wedge *cfg2* = *cfg4* \wedge

$\text{pcOf } \text{cfg3} = \text{pcOf } \text{cfg4} \wedge \text{map } \text{pcOf } \text{cfgs3} = \text{map } \text{pcOf } \text{cfgs4} \wedge$

$\text{pcOf } \text{cfg3} \in \text{PC} \wedge \text{pcOf } \text{'(set } \text{cfgs3)} \subseteq \text{PC} \wedge$

~~/// *aa1* *aa2* *aa3* *aa4* *aa5* *aa6* *aa7* *aa8* *aa9* *aa10* *aa11* *aa12* *aa13* *aa14* *aa15* *aa16* *aa17* *aa18* *aa19* *aa20* *aa21* *aa22* *aa23* *aa24* *aa25* *aa26* *aa27* *aa28* *aa29* *aa30* *aa31* *aa32* *aa33* *aa34* *aa35* *aa36* *aa37* *aa38* *aa39* *aa40* *aa41* *aa42* *aa43* *aa44* *aa45* *aa46* *aa47* *aa48* *aa49* *aa50* *aa51* *aa52* *aa53* *aa54* *aa55* *aa56* *aa57* *aa58* *aa59* *aa60* *aa61* *aa62* *aa63* *aa64* *aa65* *aa66* *aa67* *aa68* *aa69* *aa70* *aa71* *aa72* *aa73* *aa74* *aa75* *aa76* *aa77* *aa78* *aa79* *aa80* *aa81* *aa82* *aa83* *aa84* *aa85* *aa86* *aa87* *aa88* *aa89* *aa90* *aa91* *aa92* *aa93* *aa94* *aa95* *aa96* *aa97* *aa98* *aa99* *aa100* *aa101* *aa102* *aa103* *aa104* *aa105* *aa106* *aa107* *aa108* *aa109* *aa110* *aa111* *aa112* *aa113* *aa114* *aa115* *aa116* *aa117* *aa118* *aa119* *aa120* *aa121* *aa122* *aa123* *aa124* *aa125* *aa126* *aa127* *aa128* *aa129* *aa130* *aa131* *aa132* *aa133* *aa134* *aa135* *aa136* *aa137* *aa138* *aa139* *aa140* *aa141* *aa142* *aa143* *aa144* *aa145* *aa146* *aa147* *aa148* *aa149* *aa150* *aa151* *aa152* *aa153* *aa154* *aa155* *aa156* *aa157* *aa158* *aa159* *aa160* *aa161* *aa162* *aa163* *aa164* *aa165* *aa166* *aa167* *aa168* *aa169* *aa170* *aa171* *aa172* *aa173* *aa174* *aa175* *aa176* *aa177* *aa178* *aa179* *aa180* *aa181* *aa182* *aa183* *aa184* *aa185* *aa186* *aa187* *aa188* *aa189* *aa190* *aa191* *aa192* *aa193* *aa194* *aa195* *aa196* *aa197* *aa198* *aa199* *aa200* *aa201* *aa202* *aa203* *aa204* *aa205* *aa206* *aa207* *aa208* *aa209* *aa210* *aa211* *aa212* *aa213* *aa214* *aa215* *aa216* *aa217* *aa218* *aa219* *aa220* *aa221* *aa222* *aa223* *aa224* *aa225* *aa226* *aa227* *aa228* *aa229* *aa230* *aa231* *aa232* *aa233* *aa234* *aa235* *aa236* *aa237* *aa238* *aa239* *aa240* *aa241* *aa242* *aa243* *aa244* *aa245* *aa246* *aa247* *aa248* *aa249* *aa250* *aa251* *aa252* *aa253* *aa254* *aa255* *aa256* *aa257* *aa258* *aa259* *aa260* *aa261* *aa262* *aa263* *aa264* *aa265* *aa266* *aa267* *aa268* *aa269* *aa270* *aa271* *aa272* *aa273* *aa274* *aa275* *aa276* *aa277* *aa278* *aa279* *aa280* *aa281* *aa282* *aa283* *aa284* *aa285* *aa286* *aa287* *aa288* *aa289* *aa290* *aa291* *aa292* *aa293* *aa294* *aa295* *aa296* *aa297* *aa298* *aa299* *aa300* *aa301* *aa302* *aa303* *aa304* *aa305* *aa306* *aa307* *aa308* *aa309* *aa310* *aa311* *aa312* *aa313* *aa314* *aa315* *aa316* *aa317* *aa318* *aa319* *aa320* *aa321* *aa322* *aa323* *aa324* *aa325* *aa326* *aa327* *aa328* *aa329* *aa330* *aa331* *aa332* *aa333* *aa334* *aa335* *aa336* *aa337* *aa338* *aa339* *aa340* *aa341* *aa342* *aa343* *aa344* *aa345* *aa346* *aa347* *aa348* *aa349* *aa350* *aa351* *aa352* *aa353* *aa354* *aa355* *aa356* *aa357* *aa358* *aa359* *aa360* *aa361* *aa362* *aa363* *aa364* *aa365* *aa366* *aa367* *aa368* *aa369* *aa370* *aa371* *aa372* *aa373* *aa374* *aa375* *aa376* *aa377* *aa378* *aa379* *aa380* *aa381* *aa382* *aa383* *aa384* *aa385* *aa386* *aa387* *aa388* *aa389* *aa390* *aa391* *aa392* *aa393* *aa394* *aa395* *aa396* *aa397* *aa398* *aa399* *aa400* *aa401* *aa402* *aa403* *aa404* *aa405* *aa406* *aa407* *aa408* *aa409* *aa410* *aa411* *aa412* *aa413* *aa414* *aa415* *aa416* *aa417* *aa418* *aa419* *aa420* *aa421* *aa422* *aa423* *aa424* *aa425* *aa426* *aa427* *aa428* *aa429* *aa430* *aa431* *aa432* *aa433* *aa434* *aa435* *aa436* *aa437* *aa438* *aa439* *aa440* *aa441* *aa442* *aa443* *aa444* *aa445* *aa446* *aa447* *aa448* *aa449* *aa450* *aa451* *aa452* *aa453* *aa454* *aa455* *aa456* *aa457* *aa458* *aa459* *aa460* *aa461* *aa462* *aa463* *aa464* *aa465* *aa466* *aa467* *aa468* *aa469* *aa470* *aa471* *aa472* *aa473* *aa474* *aa475* *aa476* *aa477* *aa478* *aa479* *aa480* *aa481* *aa482* *aa483* *aa484* *aa485* *aa486* *aa487* *aa488* *aa489* *aa490* *aa491* *aa492* *aa493* *aa494* *aa495* *aa496* *aa497* *aa498* *aa499* *aa500* *aa501* *aa502* *aa503* *aa504* *aa505* *aa506* *aa507* *aa508* *aa509* *aa510* *aa511* *aa512* *aa513* *aa514* *aa515* *aa516* *aa517* *aa518* *aa519* *aa520* *aa521* *aa522* *aa523* *aa524* *aa525* *aa526* *aa527* *aa528* *aa529* *aa530* *aa531* *aa532* *aa533* *aa534* *aa535* *aa536* *aa537* *aa538* *aa539* *aa540* *aa541* *aa542* *aa543* *aa544* *aa545* *aa546* *aa547* *aa548* *aa549* *aa550* *aa551* *aa552* *aa553* *aa554* *aa555* *aa556* *aa557* *aa558* *aa559* *aa560* *aa561* *aa562* *aa563* *aa564* *aa565* *aa566* *aa567* *aa568* *aa569* *aa570* *aa571* *aa572* *aa573* *aa574* *aa575* *aa576* *aa577* *aa578* *aa579* *aa580* *aa581* *aa582* *aa583* *aa584* *aa585* *aa586* *aa587* *aa588* *aa589* *aa590* *aa591* *aa592* *aa593* *aa594* *aa595* *aa596* *aa597* *aa598* *aa599* *aa600* *aa601* *aa602* *aa603* *aa604* *aa605* *aa606* *aa607* *aa608* *aa609* *aa610* *aa611* *aa612* *aa613* *aa614* *aa615* *aa616* *aa617* *aa618* *aa619* *aa620* *aa621* *aa622* *aa623* *aa624* *aa625* *aa626* *aa627* *aa628* *aa629* *aa630* *aa631* *aa632* *aa633* *aa634* *aa635* *aa636* *aa637* *aa638* *aa639* *aa640* *aa641* *aa642* *aa643* *aa644* *aa645* *aa646* *aa647* *aa648* *aa649* *aa650* *aa651* *aa652* *aa653* *aa654* *aa655* *aa656* *aa657* *aa658* *aa659* *aa660* *aa661* *aa662* *aa663* *aa664* *aa665* *aa666* *aa667* *aa668* *aa669* *aa670* *aa671* *aa672* *aa673* *aa674* *aa675* *aa676* *aa677* *aa678* *aa679* *aa680* *aa681* *aa682* *aa683* *aa684* *aa685* *aa686* *aa687* *aa688* *aa689* *aa690* *aa691* *aa692* *aa693* *aa694* *aa695* *aa696* *aa697* *aa698* *aa699* *aa700* *aa701* *aa702* *aa703* *aa704* *aa705* *aa706* *aa707* *aa708* *aa709* *aa710* *aa711* *aa712* *aa713* *aa714* *aa715* *aa716* *aa717* *aa718* *aa719* *aa720* *aa721* *aa722* *aa723* *aa724* *aa725* *aa726* *aa727* *aa728* *aa729* *aa730* *aa731* *aa732* *aa733* *aa734* *aa735* *aa736* *aa737* *aa738* *aa739* *aa740* *aa741* *aa742* *aa743* *aa744* *aa745* *aa746* *aa747* *aa748* *aa749* *aa750* *aa751* *aa752* *aa753* *aa754* *aa755* *aa756* *aa757* *aa758* *aa759* *aa760* *aa761* *aa762* *aa763* *aa764* *aa765* *aa766* *aa767* *aa768* *aa769* *aa770* *aa771* *aa772* *aa773* *aa774* *aa775* *aa776* *aa777* *aa778* *aa779* *aa780* *aa781* *aa782* *aa783* *aa784* *aa785* *aa786* *aa787* *aa788* *aa789* *aa790* *aa791* *aa792* *aa793* *aa794* *aa795* *aa796* *aa797* *aa798* *aa799* *aa800* *aa801* *aa802* *aa803* *aa804* *aa805* *aa806* *aa807* *aa808* *aa809* *aa810* *aa811* *aa812* *aa813* *aa814* *aa815* *aa816* *aa817* *aa818* *aa819* *aa820* *aa821* *aa822* *aa823* *aa824* *aa825* *aa826* *aa827* *aa828* *aa829* *aa830* *aa831* *aa832* *aa833* *aa834* *aa835* *aa836* *aa837* *aa838* *aa839* *aa840* *aa841* *aa842* *aa843* *aa844* *aa845* *aa846* *aa847* *aa848* *aa849* *aa850* *aa851* *aa852* *aa853* *aa854* *aa855* *aa856* *aa857* *aa858* *aa859* *aa860* *aa861* *aa862* *aa863* *aa864* *aa865* *aa866* *aa867* *aa868* *aa869* *aa870* *aa871* *aa872* *aa873* *aa874* *aa875* *aa876* *aa877* *aa878* *aa879* *aa880* *aa881* *aa882* *aa883* *aa884* *aa885* *aa886* *aa887* *aa888* *aa889* *aa890* *aa891* *aa892* *aa893* *aa894* *aa895* *aa896* *aa897* *aa898* *aa899* *aa900* *aa901* *aa902* *aa903* *aa904* *aa905* *aa906* *aa907* *aa908* *aa909* *aa910* *aa911* *aa912* *aa913* *aa914* *aa915* *aa916* *aa917* *aa918* *aa919* *aa920* *aa921* *aa922* *aa923* *aa924* *aa925* *aa926* *aa927* *aa928* *aa929* *aa930* *aa931* *aa932* *aa933* *aa934* *aa935* *aa936* *aa937* *aa938* *aa939* *aa940* *aa941* *aa942* *aa943* *aa944* *aa945* *aa946* *aa947* *aa948* *aa949* *aa950* *aa951* *aa952* *aa953* *aa954* *aa955* *aa956* *aa957* *aa958* *aa959* *aa960* *aa961* *aa962* *aa963* *aa964* *aa965* *aa966* *aa967* *aa968* *aa969* *aa970* *aa971* *aa972* *aa973* *aa974* *aa975* *aa976* *aa977* *aa978* *aa979* *aa980* *aa981* *aa982* *aa983* *aa984* *aa985* *aa986* *aa987* *aa988* *aa989* *aa990* *aa991* *aa992* *aa993* *aa994* *aa995* *aa996* *aa997* *aa998* *aa999* *aa1000* *aa1001* *aa1002* *aa1003* *aa1004* *aa1005* *aa1006* *aa1007* *aa1008* *aa1009* *aa1010* *aa1011* *aa1012* *aa1013* *aa1014* *aa1015* *aa1016* *aa1017* *aa1018* *aa1019* *aa1020* *aa1021* *aa1022* *aa1023* *aa1024* *aa1025* *aa1026* *aa1027* *aa1028* *aa1029* *aa1030* *aa1031* *aa1032* *aa1033* *aa1034* *aa1035* *aa1036* *aa1037* *aa1038* *aa1039* *aa1040* *aa1041* *aa1042* *aa1043* *aa1044* *aa1045* *aa1046* *aa1047* *aa1048* *aa1049* *aa1050* *aa1051* *aa1052* *aa1053* *aa1054* *aa1055* *aa1056* *aa1057* *aa1058* *aa1059* *aa1060* *aa1061* *aa1062* *aa1063* *aa1064* *aa1065* *aa1066* *aa1067* *aa1068* *aa1069* *aa1070* *aa1071* *aa1072* *aa1073* *aa1074* *aa1075* *aa1076* *aa1077* *aa1078* *aa1079* *aa1080* *aa1081* *aa1082* *aa1083* *aa1084* *aa1085* *aa1086* *aa1087* *aa1088* *aa1089* *aa1090* *aa1091* *aa1092* *aa1093* *aa1094* *aa1095* *aa1096* *aa1097* *aa1098* *aa1099* *aa1100* *aa1101* *aa1102* *aa1103* *aa1104* *aa1105* *aa1106* *aa1107* *aa1108* *aa1109* *aa1110* *aa1111*~~


```

(getAvstore (stateOf cfg4))) ∧
(statA = Diff → statO = Diff)
))

```

lemmas *common-defs = common-def common-memory-def*

lemma *common-implies: common num*

```

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO ⇒
pcOf cfg1 < 9 ∧ pcOf cfg3 < 9 ∧

```

```

(n ≥ 0 → array-loc aa1 0 (getAvstore (stateOf cfg2)) ≠ array-loc aa2 n (getAvstore
(stateOf cfg2))) ∧
array-loc aa1 0 (getAvstore (stateOf cfg1)) ≠ array-loc aa2 n (getAvstore (stateOf
cfg1)))
⟨proof⟩

```

definition $\Delta 0 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

```

Δ 0 = (λnum (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO.
(common num (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO ∧

```

```

pcOf cfg3 ∈ beforeInput ∧
(ℓlength ibUT1 = ∞ ∧ ℓlength ibUT2 = ∞ ∧
ℓlength ibUT3 = ∞ ∧ ℓlength ibUT4 = ∞) ∧
(ℓhd ibUT3 ≥ NN ∧ (ℓhd ibUT1 = 0) ∧ ibUT1 = ibUT2
∨ ℓhd ibUT3 < NN ∧ ibUT1 = ibUT3 ∧ ibUT2 = ibUT4) ∧

```

```

cfg1 = cfg3 ∧ cfg2 = cfg4 ∧
ls1 = ls3 ∧ ls2 = ls4 ∧

```

$ls1 = \{\} \wedge ls2 = \{\} \wedge$
 $noMisSpec\ cfgs3$
 $\rangle\rangle$
lemmas $\Delta 0-defs' = \Delta 0-def\ common-defs\ PC-def\ beforeInput-def\ noMisSpec-def$

lemma $\Delta 0-def2$:

$\Delta 0\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO$
 $=$
 $(common\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$

~~$pcOf\ cfg3 \in\ beforeInput \wedge$~~
 $(llength\ ibUT1 = \infty \wedge llength\ ibUT2 = \infty \wedge$
 $llength\ ibUT3 = \infty \wedge llength\ ibUT4 = \infty) \wedge$
 $(ibUT1 \neq [] \wedge ibUT2 \neq [] \wedge ibUT3 \neq [] \wedge ibUT4 \neq []) \wedge$
 $(lhd\ ibUT3 \geq NN \wedge (lhd\ ibUT1 = 0) \wedge ibUT1 = ibUT2$
 $\vee lhd\ ibUT3 < NN \wedge ibUT1 = ibUT3 \wedge ibUT2 = ibUT4) \wedge$
 $pcOf\ cfg3 \in\ beforeInput \wedge$

~~$ibUT1 = ibUT2 \wedge ibUT3 = ibUT4$~~
 $cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge$
 $ls1 = ls3 \wedge ls2 = ls4 \wedge$
 $ls1 = \{\} \wedge ls2 = \{\} \wedge$
 $noMisSpec\ cfs3$
 \rangle
 $\langle proof \rangle$

lemmas $\Delta 0-defs = \Delta 0-def2\ common-defs\ PC-def\ beforeInput-def\ noMisSpec-def$

lemma $\Delta 0-implies$: $\Delta 0\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $(pcOf\ cfg3 = 1 \implies ibUT3 \neq LNil) \wedge$
 $(pcOf\ cfg4 = 1 \implies ibUT4 \neq LNil) \wedge$
 $pcOf\ cfg1 < 7 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $cfs3 = [] \wedge pcOf\ cfg3 < 7 \wedge$
 $cfs4 = [] \wedge pcOf\ cfg4 < 7$

<proof>

definition $\Delta 1 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 1 = (\lambda num$
 $(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common-strat1 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $pcOf\ cfg3 \in afterInput \wedge$
 $same-var-o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg3))\ xx < NN \wedge$

 $ls1 = ls3 \wedge ls2 = ls4 \wedge$
 $noMisSpec\ cfs3$
 $))$

lemmas $\Delta 1-defs = \Delta 1-def\ common-strat1-defs\ PC-def\ afterInput-def\ same-var-o-def$
 $noMisSpec-def$

lemma $\Delta 1-implies: \Delta 1\ num$

$(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \Longrightarrow$
 $pcOf\ cfg1 < 7 \wedge$
 $cfs3 = [] \wedge pcOf\ cfg3 \neq 1 \wedge pcOf\ cfg3 < 7 \wedge$
 $cfs4 = [] \wedge pcOf\ cfg4 \neq 1 \wedge pcOf\ cfg4 < 7$
<proof>

definition $\Delta 2 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 2 = (\lambda num$
 $(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$

```

    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO.
  (common-strat1
    (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 
    pcOf cfg3 = startOfThenBranch  $\wedge$ 
    pcOf cfg1 = pcOf cfg3  $\wedge$ 

    pcOf (last cfgs3) = elseBranch  $\wedge$ 
    same-var-o xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
    vstore (getVstore (stateOf cfg3)) xx < NN  $\wedge$ 
    ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 
    misSpecL1 cfgs3
  ))

```

lemmas $\Delta 2$ -defs = $\Delta 2$ -def common-strat1-defs PC-def same-var-def startOfThen-Branch-def
 misSpecL1-def elseBranch-def

lemma $\Delta 2$ -implies: $\Delta 2$ num
 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \implies
 pcOf (last cfgs3) = 6 \wedge pcOf cfg3 = 4 \wedge
 pcOf (last cfgs4) = pcOf (last cfgs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 length cfgs3 = Suc 0 \wedge
 length cfgs3 = length cfgs4
 <proof>

definition $\Delta 1'$:: enat \implies stateO \implies stateO \implies status \implies stateV \implies stateV \implies status
 \implies bool **where**

```

 $\Delta 1'$  = ( $\lambda$ num (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
  (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)

```

$statO.$
 $(common\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $///$
 $pcOf\ cfg3 \in\ afterInput \wedge$
 $same-var-o\ xx\ cfs3\ cfs4 \wedge$
 $(pcOf\ cfg1 > 2 \longrightarrow vstore\ (getVstore\ (stateOf\ cfg3))\ tt = vstore\ (getVstore$
 $(stateOf\ cfg4))\ tt) \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg3))\ xx \geq NN \wedge$
 $(pcOf\ cfg1 < 4 \longrightarrow pcOf\ cfg1 = pcOf\ cfg3 \wedge$
 $ls1 = \{\} \wedge ls2 = \{\} \wedge$
 $ls1 = ls3 \wedge ls2 = ls4) \wedge$
 $(pcOf\ cfg1 \leq 5 \longrightarrow ls1 \subseteq \{array-loc\ aa1\ 0\ (getAvstore\ (stateOf\ cfg1))\}$
 $\wedge ls1 = ls2 \wedge ls3 = ls4) \wedge$
 $(Language-Prelims.dist\ ls3\ ls4 \subseteq Language-Prelims.dist\ ls1\ ls2) \wedge$
 $(pcOf\ cfg1 \geq 4 \longrightarrow pcOf\ cfg1 \in\ inThenBranch \wedge pcOf\ cfg3 = elseBranch) \wedge$
 $same-xx-cp\ cfg1\ cfg2 \wedge$
 $vstore\ (getVstore\ (stateOf\ cfg1))\ xx = 0 \wedge$
 $ls3 \subseteq ls1 \wedge ls4 \subseteq ls2 \wedge$
 $noMisSpec\ cfs3$
 $)$
lemmas $\Delta 1'-defs = \Delta 1'-def\ common-defs\ PC-def\ afterInput-def$
 $same-var-o-def\ same-xx-cp-def\ noMisSpec-def\ inThenBranch-def\ elseBranch-def$
lemma $\Delta 1'-implies: \Delta 1'\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg1 < 7 \wedge pcOf\ cfg1 \neq Suc\ 0 \wedge$
 $pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $cfs3 = [] \wedge pcOf\ cfg3 < 7 \wedge$
 $cfs4 = [] \wedge pcOf\ cfg4 < 7$
 $\langle proof \rangle$

definition $\Delta 3' :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**
 $\Delta 3' = (\lambda\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

```

    (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
    statA
    (cfg1, ibT1, ibUT1, ls1)
    (cfg2, ibT2, ibUT2, ls2)
    statO.
(common num (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO  $\wedge$ 
///
pcOf cfg3 = elseBranch  $\wedge$  cfs3  $\neq$  []  $\wedge$ 
pcOf (last cfs3)  $\in$  inThenBranch  $\wedge$ 
pcOf (last cfs4) = pcOf (last cfs3)  $\wedge$ 
we/say/cp/pracs/wm/m/speculation/cfs3/cfs4
pcOf cfg1 = pcOf (last cfs3)  $\wedge$ 

same-var-o xx cfg3 cfs3 cfg4 cfs4  $\wedge$ 
(getAvstore (stateOf cfg3)) = (getAvstore (stateOf (last cfs3)))  $\wedge$ 
(getAvstore (stateOf cfg4)) = (getAvstore (stateOf (last cfs4)))  $\wedge$ 

same-xx-cp cfg1 cfg2  $\wedge$ 
ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 

vstore (getVstore (stateOf cfg3)) tt = vstore (getVstore (stateOf cfg4)) tt  $\wedge$ 

vstore (getVstore (stateOf cfg3)) xx  $\geq$  NN  $\wedge$ 

(pcOf cfg1 = 4  $\longrightarrow$  ls1 = {}  $\wedge$  ls2 = {})  $\wedge$ 
(pcOf cfg1  $\leq$  5  $\longrightarrow$  ls1  $\subseteq$  {array-loc aa1 0 (getAvstore (stateOf cfg1))}
 $\wedge$  ls2  $\subseteq$  {array-loc aa1 0 (getAvstore (stateOf cfg2))}
 $\wedge$  ls3 = ls4)  $\wedge$ 

(pcOf cfg1 > 4  $\longrightarrow$  same-var vv cfg1 (last cfs3)  $\wedge$  same-var vv cfg2 (last cfs4))
 $\wedge$ 
misSpecL1 cfs3
))
lemmas  $\Delta 3'$ -defs =  $\Delta 3'$ -def common-defs PC-def elseBranch-def
inThenBranch-def startOfThenBranch-def
same-var-o-def same-xx-cp-def misSpecL1-def same-var-def

lemma  $\Delta 3'$ -implies:  $\Delta 3'$  num (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO  $\implies$ 

```

$pcOf\ cf g1 < 7 \wedge pcOf\ cf g1 \neq Suc\ 0 \wedge$
 $pcOf\ cf g2 = pcOf\ cf g1 \wedge$
 $pcOf\ cf g3 < 7 \wedge pcOf\ cf g4 < 7 \wedge$
 $pcOf\ (last\ cf gs3) = pcOf\ (last\ cf gs4) \wedge$
 $(pcOf\ (last\ cf gs3) = 4 \vee pcOf\ (last\ cf gs3) = 5 \vee pcOf\ (last\ cf gs3) = 6) \wedge pcOf\ cf g3 = 6 \wedge$
 $pcOf\ cf g3 = pcOf\ cf g4 \wedge$
 $length\ cf gs3 = Suc\ 0 \wedge$
 $length\ cf gs3 = length\ cf gs4$
 ⟨proof⟩

definition $\Delta e :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta e = (\lambda(num::enat)\ (pstate3,cf g3,cf gs3,ibT3,ibUT3,ls3)$
 $\ (pstate4,cf g4,cf gs4,ibT4,ibUT4,ls4)$
 $\ statA$
 $\ (cf g1,ibT1,ibUT1,ls1)$
 $\ (cf g2,ibT2,ibUT2,ls2)$
 $\ statO.$
 $((num = (endPC - pcOf\ cf g1) \vee num = \infty) \wedge$
 $\ pcOf\ cf g3 = endPC \wedge pcOf\ cf g4 = endPC \wedge cf gs3 = [] \wedge cf gs4 = [] \wedge$
 $\ pcOf\ cf g1 = endPC \wedge pcOf\ cf g2 = endPC))$

lemmas $\Delta e\text{-defs} = \Delta e\text{-def}\ common\text{-def}\ endPC$

context *array-nempty*

begin

lemma *init*: *initCond* $\Delta 0$

⟨proof⟩

lemma *step0*: *unwindIntoCond* $\Delta 0$ (*oor3* $\Delta 0$ $\Delta 1$ $\Delta 1'$)

⟨proof⟩

lemma *step1*: *unwindIntoCond* $\Delta 1$ (*oor3* $\Delta 1$ $\Delta 2$ Δe)

⟨proof⟩

lemma *step2*: *unwindIntoCond* $\Delta 2$ $\Delta 1$

⟨proof⟩

lemma *xx-le-NN[simp]*: *cfg* = *Config* *pc* (*State* (*Vstore* *vs*) *avst* *h* *p*) $\implies vs\ xx = 0 \implies vs\ xx < int\ NN$

<proof>

lemma *match12I:match12* (*oor3* $\Delta 1'$ $\Delta 3'$ Δe) *ss3 ss4 statA ss1 ss2 statO* \implies
($\exists v < n.$ *proact* (*oor3* $\Delta 1'$ $\Delta 3'$ Δe) *v ss3 ss4 statA ss1 ss2 statO*) \vee
react (*oor3* $\Delta 1'$ $\Delta 3'$ Δe) *ss3 ss4 statA ss1 ss2 statO*
<proof>

lemma *step1'*: *unwindIntoCond* $\Delta 1'$ (*oor3* $\Delta 1'$ $\Delta 3'$ Δe)
<proof>

lemma *step3'*: *unwindIntoCond* $\Delta 3'$ (*oor* $\Delta 3'$ $\Delta 1'$)
<proof>

lemma *stepe*: *unwindIntoCond* Δe Δe
<proof>

lemmas *theConds* = *step0 step1 step2*
step1' step3' stepe

proposition *rsecure*
<proof>
end
end

12 Proof of Relative Security for fun5

theory *Fun5*
imports *../Instance-IMP/Instance-Secret-IMem*
Relative-Security.Unwinding
begin

12.1 Function definition and Boilerplate

no-notation *bot* ($\langle \perp \rangle$)
consts *NN* :: *nat*
consts *SS* :: *nat*
lemma *NN*: *int* *NN* ≥ 0 **and** *SS*: *int* *SS* ≥ 0 *<proof>*

definition *aa1* :: *avname* **where** *aa1* = "*a1*"
definition *aa2* :: *avname* **where** *aa2* = "*a2*"
definition *vv* :: *avname* **where** *vv* = "*v*"
definition *xx* :: *avname* **where** *xx* = "*x*"
definition *tt* :: *avname* **where** *tt* = "*y*"
definition *temp* :: *avname* **where** *temp* = "*temp*"

lemmas *vvars-defs* = *aa1-def aa2-def vv-def xx-def tt-def temp-def*

lemma *vvars-dff*[*simp*]:

aa1 ≠ *aa2* *aa1* ≠ *vv* *aa1* ≠ *xx* *aa1* ≠ *temp* *aa1* ≠ *tt*
aa2 ≠ *aa1* *aa2* ≠ *vv* *aa2* ≠ *xx* *aa2* ≠ *temp* *aa2* ≠ *tt*
vv ≠ *aa1* *vv* ≠ *aa2* *vv* ≠ *xx* *vv* ≠ *temp* *vv* ≠ *tt*
xx ≠ *aa1* *xx* ≠ *aa2* *xx* ≠ *vv* *xx* ≠ *temp* *xx* ≠ *tt*
tt ≠ *aa1* *tt* ≠ *aa2* *tt* ≠ *vv* *tt* ≠ *temp* *tt* ≠ *xx*
temp ≠ *aa1* *temp* ≠ *aa2* *temp* ≠ *vv* *temp* ≠ *xx* *temp* ≠ *tt*
{*proof*}

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

fun *initAvstore* :: *avstore* ⇒ *bool* **where**

initAvstore (*Avstore as*) = (*as aa1* = (0, *size-aa1*) ∧ *as aa2* = (*size-aa1*, *size-aa2*))

fun *istate* :: *state* ⇒ *bool* **where**

istate *s* = (*initAvstore* (*getAvstore s*))

definition *prog* ≡

[
 ~~/~~ *Start* ,
 ~~/~~ *tt* ::= (*N 0*),
 ~~/~~ *xx* ::= (*N 1*),
 ~~/~~ *IfJump* (*Not* (*Eq* (*V xx*) (*N 0*))) 4 11 ,
 ~~/~~ *Input* *U xx* ,
 ~~/~~ *IfJump* (*Less* (*V xx*) (*N NN*)) 6 10 ,
 ~~/~~ *vv* ::= *VA aa1* (*V xx*) ,
 ~~/~~ *Fence* ,
 ~~/~~ *tt* ::= (*VA aa2* (*Times* (*V vv*) (*N SS*))) ,
 ~~/~~ *Output* *U* (*V tt*) ,
 ~~/~~ *Jump* 3,
 ~~/~~ *Output* *U* (*N 0*)
]

definition *PC* ≡ {0..11}

definition *beforeWhile* = {0,1,2}

definition *inWhile* = {3..11}

definition *startOfWhileThen* = 4

definition *startOfIfThen* = 6

definition *inThenIfBeforeFence* = {6,7}

definition *startOfElseBranch* = 10

definition *inElseIf* = {10,3,4,11}

definition *whileElse* = 11

```

fun leftWhileSpec where
  leftWhileSpec cfg cfg' =
    (pcOf cfg = whileElse  $\wedge$ 
     pcOf cfg' = startOfWhileThen)

fun rightWhileSpec where
  rightWhileSpec cfg cfg' =
    (pcOf cfg = startOfWhileThen  $\wedge$ 
     pcOf cfg' = whileElse)

fun whileSpeculation where
  whileSpeculation cfg cfg' =
    (leftWhileSpec cfg cfg'  $\vee$ 
     rightWhileSpec cfg cfg')
lemmas whileSpec-def = whileSpeculation.simps
        startOfWhileThen-def
        whileElse-def

lemmas whileSpec-defs = whileSpec-def
        leftWhileSpec.simps
        rightWhileSpec.simps

lemma cases-12: (i::pcounter) = 0  $\vee$  i = 1  $\vee$  i = 2  $\vee$  i = 3  $\vee$  i = 4  $\vee$  i = 5  $\vee$ 
  i = 6  $\vee$  i = 7  $\vee$  i = 8  $\vee$  i = 9  $\vee$  i = 10  $\vee$  i = 11  $\vee$  i = 12  $\vee$  i > 12
  <proof>

lemma xx-0-cases: vs xx = 0  $\vee$  vs xx  $\neq$  0 <proof>

lemma xx-NN-cases: vs xx < int NN  $\vee$  vs xx  $\geq$  int NN <proof>

lemma is-IfJump-pcOf[simp]:
  pcOf cfg < 12  $\implies$  is-IfJump (prog ! (pcOf cfg))  $\longleftrightarrow$  pcOf cfg = 3  $\vee$  pcOf cfg = 5
  <proof>

lemma is-IfJump-pc[simp]:
  pc < 12  $\implies$  is-IfJump (prog ! pc)  $\longleftrightarrow$  pc = 3  $\vee$  pc = 5
  <proof>

lemma eq-Fence-pc[simp]:
  pc < 12  $\implies$  prog ! pc = Fence  $\longleftrightarrow$  pc = 7
  <proof>

lemma output1[simp]:
  prog ! 9 = Output U (V tt) <proof>
lemma output2[simp]:
  prog ! 11 = Output U (N 0) <proof>

```

lemma *is-if*[simp]:*is-IfJump* (prog ! 3) <proof>

lemma *is-nif1*[simp]: \neg *is-IfJump* (prog ! 6) <proof>

lemma *is-nif2*[simp]: \neg *is-IfJump* (prog ! 7) <proof>

lemma *is-nin1*[simp]: \neg *is-getInput* (prog ! 6) <proof>

lemma *is-nout1*[simp]: \neg *is-Output* (prog ! 6) <proof>

lemma *is-nin2*[simp]: \neg *is-getInput* (prog ! 10) <proof>

lemma *is-nout2*[simp]: \neg *is-Output* (prog ! 10) <proof>

lemma *fence*[simp]:prog ! 7 = *Fence* <proof>

lemma *nfence*[simp]:prog ! 6 \neq *Fence* <proof>

consts *mispred* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool*

consts *resolve* :: *predState* \Rightarrow *pcounter list* \Rightarrow *bool*

consts *update* :: *predState* \Rightarrow *pcounter list* \Rightarrow *predState*

consts *initPstate* :: *predState*

interpretation *Prog-Mispred-Init* **where**

prog = *prog* **and** *initPstate* = *initPstate* **and**

mispred = *mispred* **and** *resolve* = *resolve* **and** *update* = *update* **and**

istate = *istate*

<proof>

abbreviation

stepB-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow

bool (**infix** $\langle \rightarrow B \rangle$ 55)

where $x \rightarrow B y == \text{stepB } x y$

abbreviation

stepsB-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow

bool (**infix** $\langle \rightarrow B^* \rangle$ 55)

where $x \rightarrow B^* y == \text{star stepB } x y$

abbreviation

stepM-abbrev :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist* \Rightarrow

bool (**infix** $\langle \rightarrow MM \rangle$ 55)

where $x \rightarrow MM y == \text{stepM } x y$

abbreviation

stepN-abbrev :: *config* \times *val llist* \times *val llist* \times *loc set* \Rightarrow *config* \times *val llist* \times *val*

$l\text{list} \times \text{loc set} \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == \text{step}N x y$

abbreviation

$\text{steps}N\text{-abbrev} :: \text{config} \times \text{val llist} \times \text{val llist} \times \text{loc set} \Rightarrow \text{config} \times \text{val llist} \times \text{val llist} \times \text{loc set} \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow N^* \rangle$ 55)
where $x \rightarrow N^* y == \text{star step}N x y$

abbreviation

$\text{step}S\text{-abbrev} :: \text{config}S \Rightarrow \text{config}S \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == \text{step}S x y$

abbreviation

$\text{steps}S\text{-abbrev} :: \text{config}S \Rightarrow \text{config}S \Rightarrow \text{bool}$ (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == \text{star step}S x y$

lemma $\text{end}PC[\text{simp}]$: $\text{end}PC = 12$
 $\langle \text{proof} \rangle$

lemma $\text{is-getInput-pcOf}[\text{simp}]$: $\text{pcOf } \text{cfg} < 12 \Longrightarrow \text{is-getInput } (\text{prog}!(\text{pcOf } \text{cfg})) \longleftrightarrow \text{pcOf } \text{cfg} = 4$
 $\langle \text{proof} \rangle$

lemma $\text{getUntrustedInput-pcOf}[\text{simp}]$: $\text{prog}!4 = \text{Input } U \text{ } xx$
 $\langle \text{proof} \rangle$

lemma $\text{getInput-not6}[\text{simp}]$: $\neg \text{is-getInput } (\text{prog} ! 6)$ $\langle \text{proof} \rangle$

lemma $\text{getInput-not7}[\text{simp}]$: $\neg \text{is-getInput } (\text{prog} ! 7)$ $\langle \text{proof} \rangle$

lemma $\text{getInput-not10}[\text{simp}]$: $\neg \text{is-getInput } (\text{prog} ! 10)$ $\langle \text{proof} \rangle$

lemma $\text{is-Output-pcOf}[\text{simp}]$: $\text{pcOf } \text{cfg} < 12 \Longrightarrow \text{is-Output } (\text{prog}!(\text{pcOf } \text{cfg})) \longleftrightarrow (\text{pcOf } \text{cfg} = 9 \vee \text{pcOf } \text{cfg} = 11)$
 $\langle \text{proof} \rangle$

lemma is-Output : $\text{is-Output } (\text{prog} ! 9)$
 $\langle \text{proof} \rangle$

lemma is-Output-1 : $\text{is-Output } (\text{prog} ! 11)$
 $\langle \text{proof} \rangle$

lemma $\text{isSec}V\text{-pcOf}[\text{simp}]$:
 $\text{isSec}V (\text{cfg}, \text{ib}T, \text{ib}UT) \longleftrightarrow \text{pcOf } \text{cfg} = 0$
 $\langle \text{proof} \rangle$

lemma $\text{isSec}O\text{-pcOf}[\text{simp}]$:

$isSecO (pstate, cfg, cfgs, ibT, ibUT, ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfgs = [])$
 ⟨proof⟩

lemma $getInputT-not[simp]$: $pcOf\ cfg < 12 \implies$
 $(prog\ !\ pcOf\ cfg) \neq Input\ T\ inp$
 ⟨proof⟩

lemma $getActV-pcOf[simp]$:
 $pcOf\ cfg < 12 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 4\ then\ lhd\ ibUT\ else\ \perp)$
 ⟨proof⟩

lemma $getObsV-pcOf[simp]$:
 $pcOf\ cfg < 12 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 9 \vee pcOf\ cfg = 11\ then$
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $else\ \perp$
 $)$
 ⟨proof⟩

lemma $getActO-pcOf[simp]$:
 $pcOf\ cfg < 12 \implies$
 $getActO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 4 \wedge cfgs = []\ then\ lhd\ ibUT\ else\ \perp)$
 ⟨proof⟩

lemma $getObsO-pcOf[simp]$:
 $pcOf\ cfg < 12 \implies$
 $getObsO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 9 \vee pcOf\ cfg = 11) \wedge cfgs = []\ then$
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $else\ \perp$
 $)$
 ⟨proof⟩

lemma $eqSec-pcOf[simp]$:
 $eqSec\ (cfg1, ibT1, ibUT1, ls1)\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfgs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
 ⟨proof⟩

lemma $getActInput:pc4 = 4 \implies pc3 = 4 \implies cfgs3 = [] \implies cfgs4 = [] \implies$

$$\begin{aligned}
& \text{getActO} (pstate3, \text{Config } pc3 \text{ (State (Vstore } vs3) \text{ avst3 } h3 \text{ } p3), [], \text{ibT3,ibUT3,} \\
& ls3) = \\
& \text{getActO} (pstate4, \text{Config } pc4 \text{ (State (Vstore } vs4) \text{ avst4 } h4 \text{ } p4), [], \text{ibT4,ibUT4,} \\
& ls4) \\
& \implies \text{lhs ibUT3} = \text{lhs ibUT4} \\
& \langle \text{proof} \rangle
\end{aligned}$$

lemma *nextB-pc0[simp]*:
 $\text{nextB} (\text{Config } 0 \text{ } s, \text{ibT,ibUT}) =$
 $(\text{Config } 1 \text{ } s, \text{ibT,ibUT})$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc0[simp]*:
 $\text{readLocs} (\text{Config } 0 \text{ } s) = \{\}$
 $\langle \text{proof} \rangle$

lemma *nextB-pc1[simp]*:
 $\text{nextB} (\text{Config } 1 \text{ (State (Vstore } vs) \text{ avst } hh \text{ } p), \text{ibT,ibUT}) =$
 $((\text{Config } 2 \text{ (State (Vstore (vs(tt := 0))) avst } hh \text{ } p)), \text{ibT,ibUT})$
 $\langle \text{proof} \rangle$

lemma *nextB-pc1'[simp]*:
 $\text{nextB} (\text{Config} (\text{Suc } 0) \text{ (State (Vstore } vs) \text{ avst } hh \text{ } p), \text{ibT,ibUT}) =$
 $((\text{Config } 2 \text{ (State (Vstore (vs(tt := 0))) avst } hh \text{ } p)), \text{ibT,ibUT})$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc1[simp]*:
 $\text{readLocs} (\text{Config } 1 \text{ } s) = \{\}$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc1'[simp]*:
 $\text{readLocs} (\text{Config} (\text{Suc } 0) \text{ } s) = \{\}$
 $\langle \text{proof} \rangle$

lemma *nextB-pc2[simp]*:
 $\text{nextB} (\text{Config } 2 \text{ (State (Vstore } vs) \text{ avst } hh \text{ } p), \text{ibT,ibUT}) =$
 $((\text{Config } 3 \text{ (State (Vstore (vs(xx := 1))) avst } hh \text{ } p)), \text{ibT,ibUT})$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc2[simp]*:
 $\text{readLocs} (\text{Config } 2 \text{ } s) = \{\}$
 $\langle \text{proof} \rangle$

lemma *nextB-pc3-then[simp]*:

$vs\ xx \neq 0 \implies$
 $nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc3-else[simp]*:

$vs\ xx = 0 \implies$
 $nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 11\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc3*:

$nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ xx \neq 0\ then\ 4\ else\ 11)\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc3[simp]*:

$readLocs\ (Config\ 3\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextM-pc3-then[simp]*:

$vs\ xx = 0 \implies$
 $nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextM-pc3-else[simp]*:

$vs\ xx \neq 0 \implies$
 $nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 11\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextM-pc3*:

$nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ xx \neq 0\ then\ 11\ else\ 4)\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc4[simp]*:

$ibUT \neq LNil \implies nextB\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 5\ (State\ (Vstore\ (vs(xx := lhd\ ibUT)))\ avst\ hh\ p),\ ibT,\ ltl\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc4[simp]*:

$readLocs\ (Config\ 4\ s) = \{\}$

$\langle \text{proof} \rangle$

lemma *nextB-pc5-then[simp]*:

$vs\ xx < int\ NN \implies$

$nextB\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ 6\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *nextB-pc5-else[simp]*:

$vs\ xx \geq int\ NN \implies$

$nextB\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ 10\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *nextB-pc5*:

$nextB\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ (if\ vs\ xx < NN\ then\ 6\ else\ 10)\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc5[simp]*:

$readLocs\ (Config\ 5\ s) = \{\}$
 $\langle \text{proof} \rangle$

lemma *nextM-pc5-then[simp]*:

$vs\ xx \geq int\ NN \implies$

$nextM\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ 6\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *nextM-pc5-else[simp]*:

$vs\ xx < int\ NN \implies$

$nextM\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ 10\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *nextM-pc5*:

$nextM\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT) =$
 $(Config\ (if\ vs\ xx < NN\ then\ 10\ else\ 6)\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *nextB-pc6[simp]*:

$nextB\ (Config\ 6\ (State\ (Vstore\ vs)\ avst\ (Heap\ hh)\ p),\ ibT,ibUT) =$
 $(let\ l = array-loc\ aa1\ (nat\ (vs\ xx))\ avst$
 $in\ (Config\ 7\ (State\ (Vstore\ (vs\ (vv := hh\ l)))\ avst\ (Heap\ hh)\ p),\ ibT,ibUT)$
 $\langle \text{proof} \rangle$

lemma *readLocs-pc6*[simp]:
readLocs (*Config 6* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa1* (*nat (vs xx)*)
avst}
⟨*proof*⟩

lemma *nextB-pc7*[simp]:
nextB (*Config 7 s, ibT,ibUT*) = (*Config 8 s, ibT,ibUT*)
⟨*proof*⟩

lemma *readLocs-pc7*[simp]:
readLocs (*Config 7 s*) = {}
⟨*proof*⟩

lemma *nextB-pc8*[simp]:
nextB (*Config 8* (*State* (*Vstore vs*) *avst (Heap hh) p*), *ibT,ibUT*) =
(*let l = array-loc aa2* (*nat (vs vv * SS)*) *avst*
in (*Config 9* (*State* (*Vstore (vs(tt := hh l))*) *avst (Heap hh) p*)), *ibT,ibUT*)
⟨*proof*⟩

lemma *readLocs-pc8*[simp]:
readLocs (*Config 8* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa2* (*nat (vs vv * SS)*)
avst}
⟨*proof*⟩

lemma *nextB-pc9*[simp]:
nextB (*Config 9 s, ibT,ibUT*) = (*Config 10 s, ibT,ibUT*)
⟨*proof*⟩

lemma *readLocs-pc9*[simp]:
readLocs (*Config 9 s*) = {}
⟨*proof*⟩

lemma *nextB-pc10*[simp]:
nextB (*Config 10 s, ibT,ibUT*) = (*Config 3 s, ibT,ibUT*)
⟨*proof*⟩

lemma *readLocs-pc10*[simp]:
readLocs (*Config 10 s*) = {}
⟨*proof*⟩

lemma *nextB-pc11*[simp]:
 $nextB (Config\ 11\ s,\ ibT,\ ibUT) =$
 $(Config\ 12\ s,\ ibT,\ ibUT)$
 ⟨proof⟩

lemma *readLocs-pc11*[simp]:
 $readLocs (Config\ 11\ s) = \{\}$
 ⟨proof⟩

lemma *map-L1:length* $cfgs = Suc\ 0 \implies$
 $pcOf (last\ cfgs) = y \implies map\ pcOf\ cfgs = [y]$
 ⟨proof⟩

lemma *map-L2:length* $cfgs = 2 \implies$
 $pcOf (cfgs\ !\ 0) = x \implies$
 $pcOf (last\ cfgs) = y \implies map\ pcOf\ cfgs = [x,y]$
 ⟨proof⟩

lemma *length2-butlast:length* $cfgs = 2 \implies (cfgs\ !\ 0) = last (butlast\ cfgs)$
 ⟨proof⟩

lemma *nextB-stepB-pc*:
 $pc < 12 \implies (pc = 4 \longrightarrow ibUT \neq LNil) \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ nextB (Config\ pc\ s,\ ibT,\ ibUT)$
 ⟨proof⟩

lemma *not-finalB*:
 $pc < 12 \implies (pc = 4 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s,\ ibT,\ ibUT)$
 ⟨proof⟩

lemma *finalB-pc-iff'*:
 $pc < 12 \implies$
 $finalB (Config\ pc\ s,\ ibT,\ ibUT) \longleftrightarrow$
 $(pc = 4 \wedge ibUT = LNil)$
 ⟨proof⟩

lemma *finalB-pc-iff*:
 $pc \leq 12 \implies$
 $finalB (Config\ pc\ s,\ ibT,\ ibUT) \longleftrightarrow$
 $(pc = 12 \vee pc = 4 \wedge ibUT = LNil)$
 ⟨proof⟩

lemma *finalB-pcOf-iff[simp]*:

$pcOf\ cfg \leq 12 \implies$

$finalB\ (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 12 \vee pcOf\ cfg = 4 \wedge ibUT = LNil)$

$\langle proof \rangle$

lemma *finalS-cond:pcOf cfg < 12 \implies noMisSpec cfgs \implies ibUT \neq LNil \implies \neg*

finalS (pstate, cfg, cfgs, ibT, ibUT, ls)

$\langle proof \rangle$

lemma *finalS-cond':pcOf cfg < 12 \implies cfgs = [] \implies ibUT \neq LNil \implies \neg finalS*

(pstate, cfg, cfgs, ibT, ibUT, ls)

$\langle proof \rangle$

lemma *finalS-while-spec:*

$whileSpeculation\ cfg\ (last\ cfgs) \implies$

$length\ cfgs = Suc\ 0 \implies$

$\neg\ finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$

$\langle proof \rangle$

lemma *finalS-while-spec-L2:*

$pcOf\ cfg = 6 \implies$

$whileSpeculation\ (cfgs!0)\ (last\ cfgs) \implies$

$length\ cfgs = 2 \implies$

$\neg\ finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$

$\langle proof \rangle$

lemma *finalS-if-spec:*

$(pcOf\ (last\ cfgs) \in inThenIfBeforeFence \wedge pcOf\ cfg = 10) \vee$

$(pcOf\ (last\ cfgs) \in inElseIf \wedge pcOf\ cfg = 6) \implies$

$length\ cfgs = Suc\ 0 \implies$

$\neg\ finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$

$\langle proof \rangle$

end

12.2 Proof

theory *Fun5-secure*

imports *Fun5*

begin

definition *common* :: $enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow$

$stateV \Rightarrow status \Rightarrow bool$

where

$common = (\lambda w1 w2$

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

$statA$

$(cfg1, ibT1, ibUT1, ls1)$

$(cfg2, ibT2, ibUT2, ls2)$

$statO.$

$(pstate3 = pstate4 \wedge$

$cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge$

$pcOf\ cfg3 = pcOf\ cfg4 \wedge map\ pcOf\ cfgs3 = map\ pcOf\ cfgs4 \wedge$

$pcOf\ cfg3 \in PC \wedge pcOf\ (set\ cfgs3) \subseteq PC \wedge$

$llength\ ibUT1 = \infty \wedge llength\ ibUT2 = \infty \wedge$

$ibUT1 = ibUT3 \wedge ibUT2 = ibUT4 \wedge$

$w1 = w2 \wedge$

$///$

$array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3)) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4)) \wedge$

$(\forall\ cfg3' \in set\ cfgs3.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3')) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg3))) \wedge$

$(\forall\ cfg4' \in set\ cfgs4.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4')) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg4))) \wedge$

$array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3)) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4)) \wedge$

$(\forall\ cfg3' \in set\ cfgs3.\ array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3')) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg3))) \wedge$

$(\forall\ cfg4' \in set\ cfgs4.\ array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4')) = array\text{-}base\ aa2\ (getAvstore\ (stateOf\ cfg4))) \wedge$

$///$

$(statA = Diff \longrightarrow statO = Diff) \wedge$

$Dist\ ls1\ ls2\ ls3\ ls4))$

lemma *common-implies*: $common\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

$statA$

$(cfg1, ibT1, ibUT1, ls1)$

$(cfg2, ibT2, ibUT2, ls2)$

$statO \implies$

$pcOf\ cfg1 < 12 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$

$ibUT1 \neq [] \wedge ibUT2 \neq [] \wedge w1 = w2$

$\langle proof \rangle$

definition $\Delta 0 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta 0 = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

```

    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO.
  (common w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 
    pcOf cfg3  $\in$  beforeWhile  $\wedge$ 
    (pcOf cfg3 > 1  $\longrightarrow$  same-var-o tt cfg3 cfgs3 cfg4 cfgs4)  $\wedge$ 
    (pcOf cfg3 > 2  $\longrightarrow$  same-var-o xx cfg3 cfgs3 cfg4 cfgs4)  $\wedge$ 
    (pcOf cfg3 > 4  $\longrightarrow$  same-var-o xx cfg3 cfgs3 cfg4 cfgs4)  $\wedge$ 
    noMisSpec cfgs3
  ))

```

lemmas $\Delta 0$ -defs = $\Delta 0$ -def common-def PC-def same-var-o-def
beforeWhile-def noMisSpec-def

lemma $\Delta 0$ -implies: $\Delta 0$ num w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
(pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
statA
(cfg1,ibT1,ibUT1,ls1)
(cfg2,ibT2,ibUT2,ls2)
statO \implies
pcOf cfg1 < 12 \wedge pcOf cfg2 = pcOf cfg1 \wedge
ibUT1 \neq [] \wedge ibUT2 \neq [] \wedge cfgs3 = [] \wedge cfgs4 = []
{proof}

definition $\Delta 1$:: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV
 \Rightarrow stateV \Rightarrow status \Rightarrow bool **where**

```

 $\Delta 1$  = ( $\lambda$  num w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
  (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)
  statO.
  (common w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 
    pcOf cfg3  $\in$  inWhile  $\wedge$ 
    same-var-o xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
    noMisSpec cfgs3
  ))

```

lemmas $\Delta 1$ -defs = $\Delta 1$ -def common-def PC-def noMisSpec-def inWhile-def same-var-o-def

lemma $\Delta 1$ -implies: $\Delta 1$ n w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)

(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 pcOf cfg3 < 12 \wedge cfs3 = [] \wedge ibUT3 \neq [[]] \wedge
 pcOf cfg4 < 12 \wedge cfs4 = [] \wedge ibUT4 \neq [[]]
 <proof>

definition $\Delta 1'$:: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV
 \Rightarrow stateV \Rightarrow status \Rightarrow bool **where**

$\Delta 1'$ = (λ num w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)

(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.

(common w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)

(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge

same-var-o xx cfg3 cfs3 cfg4 cfs4 \wedge
 whileSpeculation cfg3 (last cfs3) \wedge
 misSpecL1 cfs3 \wedge misSpecL1 cfs4 \wedge
 w1 = ∞

))

lemmas $\Delta 1'$ -defs = $\Delta 1'$ -def common-def PC-def same-var-def

startOfIfThen-def startOfElseBranch-def
 misSpecL1-def whileSpec-defs

lemma $\Delta 1'$ -implies: $\Delta 1'$ num w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)

(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 pcOf cfg3 < 12 \wedge pcOf cfg4 < 12 \wedge
 whileSpeculation cfg3 (last cfs3) \wedge
 whileSpeculation cfg4 (last cfs4) \wedge
 length cfs3 = Suc 0 \wedge length cfs4 = Suc 0
 <proof>

definition $\Delta 2 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta 2 = (\lambda num\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$
 $(cfig2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$
 $(cfig2, ibT2, ibUT2, ls2)$
 $statO \wedge$

$same\text{-}var\text{-}o\ xx\ cfig3\ cfigs3\ cfig4\ cfigs4 \wedge$
 $pcOf\ cfig3 = startOfIfThen \wedge pcOf\ (last\ cfigs3) \in inElseIf \wedge$
 $misSpecL1\ cfigs3 \wedge misSpecL1\ cfigs4 \wedge$

$(pcOf\ (last\ cfigs3) = startOfElseBranch \longrightarrow w1 = \infty) \wedge$
 $(pcOf\ (last\ cfigs3) = 3 \longrightarrow w1 = 3) \wedge$

$(pcOf\ (last\ cfigs3) = startOfWhileThen \vee$
 $pcOf\ (last\ cfigs3) = whileElse \longrightarrow w1 = 1)$

)

lemmas $\Delta 2\text{-}defs = \Delta 2\text{-}def\ common\text{-}def\ PC\text{-}def\ same\text{-}var\text{-}o\text{-}def\ misSpecL1\text{-}def$
 $startOfIfThen\text{-}def\ inElseIf\text{-}def\ same\text{-}var\text{-}def$
 $startOfWhileThen\text{-}def\ whileElse\text{-}def\ startOfElseBranch\text{-}def$

lemma $\Delta 2\text{-}implies: \Delta 2\ num\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$

$(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$
 $(cfig2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ (last\ cfigs3) \in inElseIf \wedge pcOf\ cfig3 = 6 \wedge$
 $pcOf\ (last\ cfigs4) = pcOf\ (last\ cfigs3) \wedge$
 $pcOf\ cfig4 = pcOf\ cfig3 \wedge length\ cfigs3 = Suc\ 0 \wedge$
 $length\ cfigs4 = Suc\ 0 \wedge same\text{-}var\ xx\ (last\ cfigs3)\ (last\ cfigs4)$

$\langle proof \rangle$

definition $\Delta 2' :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta 2' = (\lambda num\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfig4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfig1, ibT1, ibUT1, ls1)$

```

      (cfg2,ibT2,ibUT2,ls2)
      statO.
    (common w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
      (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
      statA
      (cfg1,ibT1,ibUT1,ls1)
      (cfg2,ibT2,ibUT2,ls2)
      statO  $\wedge$ 
      same-var-o xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
      pcOf cfg3 = startOfIfThen  $\wedge$ 
      whileSpeculation (cfgs3!0) (last cfgs3)  $\wedge$ 
      misSpecL2 cfgs3  $\wedge$  misSpecL2 cfgs4  $\wedge$ 
      w1 = 2
    ))

```

lemmas $\Delta 2'$ -defs = $\Delta 2'$ -def common-def PC-def same-var-def
 startOfElseBranch-def startOfIfThen-def
 whileSpec-defs misSpecL2-def

lemma $\Delta 2'$ -implies: $\Delta 2'$ num w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \implies
 pcOf cfg3 = 6 \wedge pcOf cfg4 = 6 \wedge
 pcOf (last cfgs3) = pcOf (last cfgs4) \wedge
 whileSpeculation (cfgs3!0) (last cfgs3) \wedge
 whileSpeculation (cfgs4!0) (last cfgs4) \wedge
 length cfgs3 = 2 \wedge length cfgs4 = 2
 ⟨proof⟩

definition $\Delta 3 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV$
 $\Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

```

 $\Delta 3 = (\lambda num w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)$ 
  (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)
  statO.
  (common w1 w2 (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 
    same-var-o xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
    pcOf cfg3 = startOfElseBranch  $\wedge$  pcOf (last cfgs3)  $\in$  inThenIfBeforeFence  $\wedge$ 

```

$misSpecL1\ cfgs3 \wedge$
 $(pcOf\ (last\ cfgs3) = 6 \longrightarrow w1 = \infty) \wedge$
 $(pcOf\ (last\ cfgs3) = 7 \longrightarrow w1 = 1)$
 $)$

lemmas $\Delta3-defs = \Delta3-def\ common-def\ PC-def\ same-var-o-def$
 $startOfElseBranch-def\ inThenIfBeforeFence-def$

lemma $\Delta3-implies: \Delta3\ num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ (last\ cfgs3) \in inThenIfBeforeFence \wedge$
 $pcOf\ (last\ cfgs4) = pcOf\ (last\ cfgs3) \wedge$
 $pcOf\ cfg3 = 10 \wedge pcOf\ cfg3 = pcOf\ cfg4 \wedge$
 $length\ cfgs3 = Suc\ 0 \wedge length\ cfgs4 = Suc\ 0$
 $\langle proof \rangle$

definition $\Delta e :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow$
 $stateV \Rightarrow status \Rightarrow bool$ **where**
 $\Delta e = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(pcOf\ cfg3 = endPC \wedge pcOf\ cfg4 = endPC \wedge cfgs3 = [] \wedge cfgs4 = [] \wedge$
 $pcOf\ cfg1 = endPC \wedge pcOf\ cfg2 = endPC))$

lemmas $\Delta e-defs = \Delta e-def\ common-def\ endPC-def$

lemma $init: initCond\ \Delta 0$
 $\langle proof \rangle$

lemma $step0: unwindIntoCond\ \Delta 0\ (oor\ \Delta 0\ \Delta 1)$
 $\langle proof \rangle$

lemma $step1: unwindIntoCond\ \Delta 1\ (oor5\ \Delta 1\ \Delta 1'\ \Delta 2\ \Delta 3\ \Delta e)$

<proof>

lemma *step2: unwindIntoCond Δ_2 (oor3 Δ_2 Δ_2' Δ_1)*
<proof>

lemma *step3: unwindIntoCond Δ_3 (oor Δ_3 Δ_1)*
<proof>

lemma *step4: unwindIntoCond Δ_1' Δ_1*
<proof>

lemma *step5: unwindIntoCond Δ_2' Δ_2*
<proof>

lemma *stepe: unwindIntoCond Δ_e Δ_e*
<proof>

lemmas *theConds = step0 step1 step2 step3 step4 step5 stepe*

proposition *lrsecure*
<proof>

end

13 Proof of Relative Security for fun6

theory *Fun6*
imports *../Instance-IMP/Instance-Secret-IMem-Inp*
Relative-Security.Unwinding
begin

13.1 Function definition and Boilerplate

no-notation *bot ($\langle \perp \rangle$)*

```

consts NN :: nat
lemma NN: NN ≥ 0 <proof>

```

```

definition aa1 :: avname where aa1 = "a1"
definition aa2 :: avname where aa2 = "a2"
definition vv :: vname where vv = "v"
definition tt :: vname where tt = "y"

```

```

lemmas vvars-defs = aa1-def aa2-def vv-def xx-def tt-def yy-def ffile-def

```

```

lemma vvars-dff[simp]:
aa1 ≠ aa2 aa1 ≠ vv aa1 ≠ xx aa1 ≠ yy aa1 ≠ tt aa1 ≠ ffile
aa2 ≠ aa1 aa2 ≠ vv aa2 ≠ xx aa2 ≠ yy aa2 ≠ tt aa2 ≠ ffile
vv ≠ aa1 vv ≠ aa2 vv ≠ xx vv ≠ yy vv ≠ tt vv ≠ ffile
xx ≠ aa1 xx ≠ aa2 xx ≠ vv xx ≠ yy xx ≠ tt xx ≠ ffile
tt ≠ aa1 tt ≠ aa2 tt ≠ vv tt ≠ yy tt ≠ xx tt ≠ ffile
yy ≠ aa1 yy ≠ aa2 yy ≠ vv yy ≠ xx yy ≠ tt yy ≠ ffile
ffile ≠ aa1 ffile ≠ aa2 ffile ≠ vv ffile ≠ xx ffile ≠ tt ffile ≠ yy
<proof>

```

```

consts size-aa1 :: nat
consts size-aa2 :: nat

```

```

fun initAvstore :: avstore ⇒ bool where
initAvstore (Avstore as) = (as aa1 = (0, size-aa1) ∧ as aa2 = (size-aa1, size-aa2))

```

```

fun istate :: state ⇒ bool where
istate s = (initAvstore (getAvstore s))

```

```

definition prog ≡
[
  / Start ,
  / tt ::= (N 0),
  / xx ::= (N 1),
  / IfJump (Not (Eq (V xx) (N 0))) 4 13 ,
  / Input U xx ,
  / Input T yy ,
  / IfJump (Less (V xx) (N NN)) 7 12 ,
  / vv ::= VA aa1 (V xx) ,
  / writeSecretOnFile,
  / Fence ,
  / tt ::= (VA aa2 (Times (V vv) (N 512))) ,
  / Output U (V tt) ,
  / Jump 3,
  / Output U (N 0)
]

```

definition $PC \equiv \{0..13\}$

definition $beforeWhile = \{0,1,2\}$

definition $afterWhile = \{3..13\}$

definition $startOfWhileThen = 4$

definition $startOfIfThen = 7$

definition $inThenIfBeforeOutput = \{7,8\}$

definition $startOfElseBranch = 12$

definition $inElseIf = \{12,3,4,13\}$

definition $whileElse = 13$

fun $leftWhileSpec$ **where**

$leftWhileSpec\ cfg\ cfg' =$
 $(pcOf\ cfg = whileElse \wedge$
 $pcOf\ cfg' = startOfWhileThen)$

fun $rightWhileSpec$ **where**

$rightWhileSpec\ cfg\ cfg' =$
 $(pcOf\ cfg = startOfWhileThen \wedge$
 $pcOf\ cfg' = whileElse)$

fun $whileSpeculation$ **where**

$whileSpeculation\ cfg\ cfg' =$
 $(leftWhileSpec\ cfg\ cfg' \vee$
 $rightWhileSpec\ cfg\ cfg')$

lemmas $whileSpec-def = whileSpeculation.simps$

$startOfWhileThen-def$
 $whileElse-def$

lemmas $whileSpec-defs = whileSpec-def$

$leftWhileSpec.simps$
 $rightWhileSpec.simps$

lemma $cases-14: (i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee$
 $i = 6 \vee i = 7 \vee i = 8 \vee i = 9 \vee i = 10 \vee i = 11 \vee i = 12 \vee i = 13 \vee i = 14$
 $\vee i > 14$
 $\langle proof \rangle$

lemma $xx-0-cases: vs\ xx = 0 \vee vs\ xx \neq 0 \langle proof \rangle$

lemma $xx-NN-cases: vs\ xx < int\ NN \vee vs\ xx \geq int\ NN \langle proof \rangle$

lemma $is-If-pcOf[simp]:$

$pcOf\ cfg < 14 \implies is-IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 3 \vee pcOf\ cfg = 6$
 $\langle proof \rangle$

lemma *is-If-pc*[simp]:

$pc < 14 \implies is-IfJump (prog ! pc) \longleftrightarrow pc = 3 \vee pc = 6$
<proof>

lemma *eq-Fence-pc*[simp]:

$pc < 14 \implies prog ! pc = Fence \longleftrightarrow pc = 9$
<proof>

lemma *output1*[simp]: $prog ! 11 = Output\ U\ (V\ tt)$ *<proof>*

lemma *output2*[simp]: $prog ! 13 = Output\ U\ (N\ 0)$ *<proof>*

lemma *is-if*[simp]: $is-IfJump (prog ! 3)$ *<proof>*

lemma *is-nif1*[simp]: $\neg is-IfJump (prog ! 7)$ *<proof>*

lemma *is-nif2*[simp]: $\neg is-IfJump (prog ! 8)$ *<proof>*

lemma *getInput-not6*[simp]: $\neg is-getInput (prog ! 6)$ *<proof>*

lemma *Output-not6*[simp]: $\neg is-Output (prog ! 6)$ *<proof>*

lemma *getInput-not7*[simp]: $\neg is-getInput (prog ! 7)$ *<proof>*

lemma *Output-not7*[simp]: $\neg is-Output (prog ! 7)$ *<proof>*

lemma *getInput-not8*[simp]: $\neg is-getInput (prog ! 8)$ *<proof>*

lemma *Output-not8*[simp]: $is-Output (prog ! 8)$ *<proof>*

lemma *is-nif*[simp]: $\neg is-IfJump (prog ! 9)$ *<proof>*

lemma *getInput-not10*[simp]: $\neg is-getInput (prog ! 10)$ *<proof>*

lemma *Output-not10*[simp]: $\neg is-Output (prog ! 10)$ *<proof>*

lemma *getInput-not12*[simp]: $\neg is-getInput (prog ! 12)$ *<proof>*

lemma *Output-not12*[simp]: $\neg is-Output (prog ! 12)$ *<proof>*

lemma *fence*[simp]: $prog ! 9 = Fence$ *<proof>*

lemma *nfence*[simp]: $prog ! 7 \neq Fence$ *<proof>*

consts *mispred* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$

consts *resolve* :: $predState \Rightarrow pcounter\ list \Rightarrow bool$

consts *update* :: $predState \Rightarrow pcounter\ list \Rightarrow predState$

consts *initPstate* :: $predState$

interpretation *Prog-Mispred-Init* where

$prog = prog$ **and** $initPstate = initPstate$ **and**

$mispred = mispred$ **and** $resolve = resolve$ **and** $update = update$ **and**

$istate = istate$

<proof>

abbreviation

stepB-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** $\langle \rightarrow B \rangle$ 55)
where $x \rightarrow B y == \text{step}B\ x\ y$

abbreviation

stepsB-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** $\langle \rightarrow B^* \rangle$ 55)
where $x \rightarrow B^* y == \text{star}\ \text{step}B\ x\ y$

abbreviation

stepM-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** $\langle \rightarrow MM \rangle$ 55)
where $x \rightarrow MM y == \text{step}M\ x\ y$

abbreviation

stepN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val*
llist × *loc set* ⇒ *bool* (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == \text{step}N\ x\ y$

abbreviation

stepsN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val*
llist × *loc set* ⇒ *bool* (**infix** $\langle \rightarrow N^* \rangle$ 55)
where $x \rightarrow N^* y == \text{star}\ \text{step}N\ x\ y$

abbreviation

stepS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** $\langle \rightarrow S \rangle$ 55)
where $x \rightarrow S y == \text{step}S\ x\ y$

abbreviation

stepsS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** $\langle \rightarrow S^* \rangle$ 55)
where $x \rightarrow S^* y == \text{star}\ \text{step}S\ x\ y$

lemma *endPC[simp]*: *endPC* = 14

<proof>

lemma *is-getInput-pcOf[simp]*: *pcOf cfg* < 14 ⇒ *is-getInput* (*prog!*(*pcOf cfg*))

↔ *pcOf cfg* = 4 ∨ *pcOf cfg* = 5

<proof>

lemma *is-Output-pcOf[simp]*: *pcOf cfg* < 14 ⇒ *is-Output* (*prog!*(*pcOf cfg*)) ↔

$(pcOf\ cfg = 8 \vee pcOf\ cfg = 11 \vee pcOf\ cfg = 13)$
 $\langle proof \rangle$

lemma $is-getInput1[simp]: is-getInput\ (prog\ !\ 4)\ \langle proof \rangle$

lemma $is-Output-T: is-Output\ (prog\ !\ 8)$
 $\langle proof \rangle$

lemma $is-Output: is-Output\ (prog\ !\ 11)$
 $\langle proof \rangle$

lemma $is-Output-1: is-Output\ (prog\ !\ 13)$
 $\langle proof \rangle$

lemma $isSecV-pcOf[simp]:$
 $isSecV\ (cfg, ibT, ibUT, ls) \longleftrightarrow \neg finalB\ (cfg, ibT, ibUT)$
 $\langle proof \rangle$

lemma $isSecO-pcOf[simp]:$
 $isSecO\ (pstate, cfg, cfs, ibT, ibUT, ls) \longleftrightarrow$
 $\neg finalS\ (pstate, cfg, cfs, ibT, ibUT, ls) \wedge cfs = []$
 $\langle proof \rangle$

lemma $getActV-pcOf[simp]:$
 $pcOf\ cfg < 14 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 4\ then\ lhd\ ibUT$
 $\ else\ if\ pcOf\ cfg = 5\ then\ lhd\ ibT$
 $\ else\ \perp)$
 $\langle proof \rangle$

lemma $getObsV-pcOf[simp]:$
 $pcOf\ cfg < 14 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 11 \vee pcOf\ cfg = 13\ then$
 $\ (outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $\ else\ \perp$
 $\)$
 $\langle proof \rangle$

lemma $getActO-pcOf[simp]:$
 $pcOf\ cfg < 12 \implies$
 $getActO\ (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ cfs = []\ then$
 $\ (if\ pcOf\ cfg = 4\ then\ lhd\ ibUT$
 $\ \ else\ if\ pcOf\ cfg = 5\ then\ lhd\ ibT$
 $\ \ else\ \perp)\ else\ \perp)$
 $\langle proof \rangle$

lemma *getObsO-pcOf[simp]*:
 $pcOf\ cfg < 14 \implies$
 $getObsO\ (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 11 \vee pcOf\ cfg = 13) \wedge cfs = []\ then$
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg), ls)$
 $else\ \perp$
 $)$
 $\langle proof \rangle$

lemma *getActTrustedInput:pc4 = 4 \implies pc3 = 4 \implies cfs3 = [] \implies cfs4 = [] \implies*
 \implies
 $getActO\ (pstate3, Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3), [], ib3T,$
 $ib3UT, ls3) =$
 $getActO\ (pstate4, Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4), [], ib4T,$
 $ib4UT, ls4)$
 $\implies\ lhd\ ib3UT = lhd\ ib4UT$
 $\langle proof \rangle$

lemma *getActUntrustedInput:pc4 = 5 \implies pc3 = 5 \implies cfs3 = [] \implies cfs4 = [] \implies*
 \implies
 $getActO\ (pstate3, Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3), [], ib3T,$
 $ib3UT, ls3) =$
 $getActO\ (pstate4, Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4), [], ib4T,$
 $ib4UT, ls4)$
 $\implies\ lhd\ ib3T = lhd\ ib4T$
 $\langle proof \rangle$

lemma *nextB-pc0[simp]*:
 $nextB\ (Config\ 0\ s, ibT, ibUT) = (Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc0[simp]*:
 $readLocs\ (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc1[simp]*:
 $nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ hh\ p), ibT, ibUT) =$
 $((Config\ 2\ (State\ (Vstore\ (vs(tt := 0)))\ avst\ hh\ p)), ibT, ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc1'[simp]*:

$nextB (Config (Suc 0) (State (Vstore vs) avst hh p), ibT, ibUT) =$
 $((Config 2 (State (Vstore (vs(tt := 0)))) avst hh p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc1[simp]$:
 $readLocs (Config 1 s) = \{\}$
 $\langle proof \rangle$

lemma $readLocs-pc1'[simp]$:
 $readLocs (Config (Suc 0) s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc2[simp]$:
 $nextB (Config 2 (State (Vstore vs) avst hh p), ibT, ibUT) =$
 $((Config 3 (State (Vstore (vs(xx := 1)))) avst hh p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc2[simp]$:
 $readLocs (Config 2 s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc3-then[simp]$:
 $vs\ xx \neq 0 \implies$
 $nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =$
 $(Config 4 (State (Vstore vs) avst hh p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $nextB-pc3-else[simp]$:
 $vs\ xx = 0 \implies$
 $nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =$
 $(Config 13 (State (Vstore vs) avst hh p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $nextB-pc3$:
 $nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =$
 $(Config (if vs\ xx \neq 0 then 4 else 13) (State (Vstore vs) avst hh p), ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc3[simp]$:
 $readLocs (Config 3 s) = \{\}$
 $\langle proof \rangle$

lemma $nextM-pc3-then[simp]$:
 $vs\ xx = 0 \implies$
 $nextM (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =$

(*Config 4* (State (Vstore vs) avst hh p), ibT, ibUT)
<proof>

lemma *nextM-pc3-else*[simp]:

$vs\ xx \neq 0 \implies$
 $nextM$ (*Config 3* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config 13* (State (Vstore vs) avst hh p), ibT, ibUT)
<proof>

lemma *nextM-pc3*:

$nextM$ (*Config 3* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config* (if vs xx \neq 0 then 13 else 4) (State (Vstore vs) avst hh p), ibT, ibUT)
<proof>

lemma *nextB-pc4*[simp]:

$ibT \neq LNil \implies nextB$ (*Config 4* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config 5* (State (Vstore (vs(xx := lhd ibUT))) avst hh p), ibT, ltl ibUT)
<proof>

lemma *readLocs-pc4*[simp]:

$readLocs$ (*Config 4* s) = {}
<proof>

lemma *nextB-pc5*[simp]:

$ibT \neq LNil \implies nextB$ (*Config 5* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config 6* (State (Vstore (vs(yy := lhd ibT))) avst hh p), ltl ibT, ibUT)
<proof>

lemma *readLocs-pc5*[simp]:

$readLocs$ (*Config 5* s) = {}
<proof>

lemma *nextB-pc6-then*[simp]:

$vs\ xx < int\ NN \implies$
 $nextB$ (*Config 6* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config 7* (State (Vstore vs) avst hh p), ibT, ibUT)
<proof>

lemma *nextB-pc6-else*[simp]:

$vs\ xx \geq int\ NN \implies$
 $nextB$ (*Config 6* (State (Vstore vs) avst hh p), ibT, ibUT) =
(*Config 12* (State (Vstore vs) avst hh p), ibT, ibUT)
<proof>

lemma *nextB-pc6*:

nextB (*Config 6* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*) =
(*Config* (*if vs xx < int NN then 7 else 12*) (*State* (*Vstore vs*) *avst hh p*), *ibT*,
ibUT)
{*proof*}

lemma *readLocs-pc6[simp]*:

readLocs (*Config 6 s*) = {}
{*proof*}

lemma *nextM-pc6-then[simp]*:

vs xx ≥ int NN \implies
nextM (*Config 6* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*) =
(*Config 7* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*)
{*proof*}

lemma *nextM-pc6-else[simp]*:

vs xx < int NN \implies
nextM (*Config 6* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*) =
(*Config 12* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*)
{*proof*}

lemma *nextM-pc6*:

nextM (*Config 6* (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*) =
(*Config* (*if vs xx < int NN then 12 else 7*) (*State* (*Vstore vs*) *avst hh p*), *ibT*,
ibUT)
{*proof*}

lemma *nextB-pc7[simp]*:

nextB (*Config 7* (*State* (*Vstore vs*) *avst (Heap hh) p*), *ibT*, *ibUT*) =
(*let l = array-loc aa1 (nat (vs xx)) avst*
in (Config 8 (State (Vstore (vs(vv := hh l))) avst (Heap hh) p)), *ibT*, *ibUT*)
{*proof*}

lemma *readLocs-pc7[simp]*:

readLocs (*Config 7 (State (Vstore vs) avst hh p)*) = {*array-loc aa1 (nat (vs xx))*
avst}
{*proof*}

lemma *nextB-pc8[simp]*:

nextB (*Config 8 (State (Vstore vs) avst hh p)*, *ibT*, *ibUT*) =
(*Config 9 (State (Vstore vs) avst hh p)*), *ibT*, *ibUT*)
{*proof*}

lemma *readLocs-pc8[simp]*:

$readLocs (Config\ 8\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc9[simp]$:
 $nextB (Config\ 9\ s, ibT, ibUT) = (Config\ 10\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc9[simp]$:
 $readLocs (Config\ 9\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc10[simp]$:
 $nextB (Config\ 10\ (State\ (Vstore\ vs)\ avst\ (Heap\ hh)\ p), ibT, ibUT) =$
 $(let\ l = array-loc\ aa2\ (nat\ (vs\ vv * 512))\ avst$
 $in\ (Config\ 11\ (State\ (Vstore\ (vs(tt := hh\ l)))\ avst\ (Heap\ hh)\ p)), ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc10[simp]$:
 $readLocs (Config\ 10\ (State\ (Vstore\ vs)\ avst\ hh\ p)) = \{array-loc\ aa2\ (nat\ (vs\ vv * 512))\ avst\}$
 $\langle proof \rangle$

lemma $nextB-pc11[simp]$:
 $nextB (Config\ 11\ s, ibT, ibUT) = (Config\ 12\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc11[simp]$:
 $readLocs (Config\ 11\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc12[simp]$:
 $nextB (Config\ 12\ s, ibT, ibUT) = (Config\ 3\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc12[simp]$:
 $readLocs (Config\ 12\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc13[simp]$:
 $nextB (Config\ 13\ s, ibT, ibUT) =$

(*Config 14 s, ibT, ibUT*)
<proof>

lemma *readLocs-pc13[simp]*:
readLocs (Config 13 s) = {}
<proof>

lemma *map-L1:length cfgs = Suc 0* \implies
pcOf (last cfgs) = y \implies *map pcOf cfgs = [y]*
<proof>

lemma *map-L2:length cfgs = 2* \implies
pcOf (cfgs ! 0) = x \implies
pcOf (last cfgs) = y \implies *map pcOf cfgs = [x,y]*
<proof>

lemma *length2-butlast-eq:length cfgs = 2* \implies *(cfgs ! 0) = last (butlast cfgs)*
<proof>

lemma *nextB-stepB-pc*:
pc < 14 \implies (*pc = 4* \longrightarrow *ibUT \neq LNil*) \implies (*pc = 5* \longrightarrow *ibT \neq LNil*) \implies
(*Config pc s, ibT, ibUT*) \rightarrow *B nextB (Config pc s, ibT, ibUT)*
<proof>

lemma *not-finalB*:
pc < 14 \implies (*pc = 4* \longrightarrow *ibUT \neq LNil*) \implies (*pc = 5* \longrightarrow *ibT \neq LNil*) \implies
 \neg *finalB (Config pc s, ibT, ibUT)*
<proof>

lemma *finalB-pc-iff'*:
pc < 14 \implies
finalB (Config pc s, ibT, ibUT) \longleftrightarrow
(*pc = 4* \wedge *ibUT = LNil*) \vee (*pc = 5* \wedge *ibT = LNil*)
<proof>

lemma *finalB-pc-iff*:
pc \leq 14 \implies
finalB (Config pc s, ibT, ibUT) \longleftrightarrow
(*pc = 14* \vee (*pc = 4* \wedge *ibUT = LNil*)) \vee (*pc = 5* \wedge *ibT = LNil*)
<proof>

lemma *finalB-pcOf-iff[simp]*:
pcOf cfg \leq 14 \implies
finalB (cfg, ibT, ibUT) \longleftrightarrow (*pcOf cfg = 14* \vee (*pcOf cfg = 4* \wedge *ibUT = LNil*)) \vee

($pcOf\ cfg = 5 \wedge ibT = LNil$)
 ⟨proof⟩

lemma *finalS-cond*: $pcOf\ cfg < 14 \implies noMisSpec\ cfgs \implies ibT \neq LNil \implies ibUT \neq LNil \implies \neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *finalS-cond'*: $pcOf\ cfg < 14 \implies cfgs = [] \implies ibT \neq LNil \implies ibUT \neq LNil \implies \neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *finalS-while-spec*:
 $whileSpeculation\ cfg\ (last\ cfgs) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *finalS-while-spec-L2*:
 $pcOf\ cfg = 7 \implies$
 $whileSpeculation\ (cfgs!0)\ (last\ cfgs) \implies$
 $length\ cfgs = 2 \implies$
 $\neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *finalS-if-spec*:
 $(pcOf\ (last\ cfgs) \in inThenIfBeforeOutput \wedge pcOf\ cfg = 12) \vee$
 $(pcOf\ (last\ cfgs) \in inElseIf \wedge pcOf\ cfg = 7) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
 ⟨proof⟩

end

13.2 Proof

theory *Fun6-secure*
imports *Fun6*
begin

definition *common* :: $enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$

where

$common = (\lambda w1\ w2$
 $\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(pstate3 = pstate4 \wedge$
 $cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge$
 $pcOf\ cfg3 = pcOf\ cfg4 \wedge map\ pcOf\ cfs3 = map\ pcOf\ cfs4 \wedge$
 $pcOf\ cfg3 \in PC \wedge pcOf\ (set\ cfs3) \subseteq PC \wedge$
 $llength\ ibT1 = \infty \wedge llength\ ibT2 = \infty \wedge$
 $llength\ ibUT1 = \infty \wedge llength\ ibUT2 = \infty \wedge$

 $ibT1 = ibT3 \wedge ibT2 = ibT4 \wedge$
 $ibUT1 = ibUT3 \wedge ibUT2 = ibUT4 \wedge$

 $w1 = w2 \wedge$
 $///$
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg3)) = array-base\ aa1\ (getAvstore\ (stateOf$
 $cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfs3. array-base\ aa1\ (getAvstore\ (stateOf\ cfg3')) = array-base\ aa1$
 $(getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfs4. array-base\ aa1\ (getAvstore\ (stateOf\ cfg4')) = array-base\ aa1$
 $(getAvstore\ (stateOf\ cfg4))) \wedge$
 $array-base\ aa2\ (getAvstore\ (stateOf\ cfg3)) = array-base\ aa2\ (getAvstore\ (stateOf$
 $cfg4)) \wedge$
 $(\forall\ cfg3' \in set\ cfs3. array-base\ aa2\ (getAvstore\ (stateOf\ cfg3')) = array-base\ aa2$
 $(getAvstore\ (stateOf\ cfg3))) \wedge$
 $(\forall\ cfg4' \in set\ cfs4. array-base\ aa2\ (getAvstore\ (stateOf\ cfg4')) = array-base\ aa2$
 $(getAvstore\ (stateOf\ cfg4))) \wedge$
 $///$
 $(statA = Diff \longrightarrow statO = Diff) \wedge$
 $Dist\ ls1\ ls2\ ls3\ ls4))$

lemma *common-implies: common* $w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg1 < 14 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $ibT1 \neq [] \wedge ibT2 \neq [] \wedge$
 $ibUT1 \neq [] \wedge ibUT2 \neq [] \wedge$
 $w1 = w2$
 $\langle proof \rangle$

definition $\Delta 0 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV$
 $\Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

```

Δ0 = (λnum w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
      (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
      statA
      (cfg1, ibT1, ibUT1, ls1)
      (cfg2, ibT2, ibUT2, ls2)
      statO.
      (common w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
        (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
        statA
        (cfg1, ibT1, ibUT1, ls1)
        (cfg2, ibT2, ibUT2, ls2)
        statO ∧
        pcOf cfg3 ∈ beforeWhile ∧
        (pcOf cfg3 > 1 → same-var-o tt cfg3 cfs3 cfg4 cfs4) ∧
        (pcOf cfg3 > 2 → same-var-o xx cfg3 cfs3 cfg4 cfs4) ∧
        (pcOf cfg3 > 4 → same-var-o xx cfg3 cfs3 cfg4 cfs4) ∧
        noMisSpec cfs3
      ))

```

lemmas Δ0-defs = Δ0-def common-def PC-def same-var-o-def
beforeWhile-def noMisSpec-def

lemma Δ0-implies: Δ0 num w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO ⇒
pcOf cfg1 < 14 ∧ pcOf cfg2 = pcOf cfg1 ∧
ibT1 ≠ [] ∧ ibT2 ≠ [] ∧
ibUT1 ≠ [] ∧ ibUT2 ≠ [] ∧
cfs4 = []
(proof)

definition Δ1 :: enat ⇒ enat ⇒ enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV
⇒ stateV ⇒ status ⇒ bool **where**

```

Δ1 = (λnum w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
      (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
      statA
      (cfg1, ibT1, ibUT1, ls1)
      (cfg2, ibT2, ibUT2, ls2)
      statO.
      (common w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
        (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
        statA
        (cfg1, ibT1, ibUT1, ls1)
        (cfg2, ibT2, ibUT2, ls2)
        statO ∧

```

$pcOf\ cfg3 \in afterWhile \wedge$
 $same-var-o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $noMisSpec\ cfs3$
 $)$
lemmas $\Delta1-defs = \Delta1-def\ common-def\ noMisSpec-def\ PC-def\ afterWhile-def\ same-var-o-def$
lemma $\Delta1-implies: \Delta1\ n\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg3 < 14 \wedge cfs3 = [] \wedge ibT3 \neq [] \wedge$
 $pcOf\ cfg4 < 14 \wedge cfs4 = [] \wedge ibT4 \neq [] \wedge$
 $ibUT3 \neq [] \wedge ibUT4 \neq []$
 $\langle proof \rangle$

definition $\Delta1' :: enat \implies enat \implies enat \implies stateO \implies stateO \implies status \implies stateV$
 $\implies stateV \implies status \implies bool$ **where**
 $\Delta1' = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $same-var-o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $whileSpeculation\ cfg3\ (last\ cfs3) \wedge$
 $misSpecL1\ cfs3 \wedge misSpecL1\ cfs4 \wedge$
 $w1 = \infty$
 $)$
 $)$

lemmas $\Delta1'-defs = \Delta1'-def\ common-def\ PC-def\ same-var-def$
 $startOfIfThen-def\ startOfElseBranch-def$
 $misSpecL1-def\ whileSpec-defs$

lemma $\Delta1'-implies: \Delta1'\ num\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg3 < 14 \wedge pcOf\ cfg4 < 14 \wedge$

$whileSpeculation\ cf_3\ (last\ cfs_3) \wedge$
 $whileSpeculation\ cf_4\ (last\ cfs_4) \wedge$
 $length\ cfs_3 = Suc\ 0 \wedge length\ cfs_4 = Suc\ 0$
 ⟨proof⟩

definition $\Delta_2 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV$
 $\Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta_2 = (\lambda num\ w1\ w2\ (pstate_3, cf_3, cfs_3, ibT_3, ibUT_3, ls_3)$
 $(pstate_4, cf_4, cfs_4, ibT_4, ibUT_4, ls_4)$
 $statA$
 $(cf_1, ibT_1, ibUT_1, ls_1)$
 $(cf_2, ibT_2, ibUT_2, ls_2)$
 $statO.$
 $(common\ w1\ w2\ (pstate_3, cf_3, cfs_3, ibT_3, ibUT_3, ls_3)$
 $(pstate_4, cf_4, cfs_4, ibT_4, ibUT_4, ls_4)$
 $statA$
 $(cf_1, ibT_1, ibUT_1, ls_1)$
 $(cf_2, ibT_2, ibUT_2, ls_2)$
 $statO \wedge$

$same-var-o\ xx\ cf_3\ cfs_3\ cf_4\ cfs_4 \wedge$
 $pcOf\ cf_3 = startOfIfThen \wedge pcOf\ (last\ cfs_3) \in inElseIf \wedge$
 $misSpecL1\ cfs_3 \wedge misSpecL1\ cfs_4 \wedge$

$(pcOf\ (last\ cfs_3) = startOfElseBranch \longrightarrow w1 = \infty) \wedge$
 $(pcOf\ (last\ cfs_3) = 3 \longrightarrow w1 = 3) \wedge$

$(pcOf\ (last\ cfs_3) = startOfWhileThen \vee$
 $pcOf\ (last\ cfs_3) = whileElse \longrightarrow w1 = 1)$

))

lemmas $\Delta_2-defs = \Delta_2-def\ common-def\ PC-def\ same-var-o-def\ misSpecL1-def$
 $startOfIfThen-def\ inElseIf-def\ same-var-def$
 $startOfWhileThen-def\ whileElse-def\ startOfElseBranch-def$

lemma $\Delta_2-implies: \Delta_2\ num\ w1\ w2\ (pstate_3, cf_3, cfs_3, ibT_3, ibUT_3, ls_3)$

$(pstate_4, cf_4, cfs_4, ibT_4, ibUT_4, ls_4)$
 $statA$
 $(cf_1, ibT_1, ibUT_1, ls_1)$
 $(cf_2, ibT_2, ibUT_2, ls_2)$
 $statO \implies$

$pcOf\ (last\ cfs_3) \in inElseIf \wedge pcOf\ cf_3 = 7 \wedge$

$pcOf\ (last\ cfs_4) = pcOf\ (last\ cfs_3) \wedge$

$pcOf\ cf_4 = pcOf\ cf_3 \wedge length\ cfs_3 = Suc\ 0 \wedge$

$length\ cfs_4 = Suc\ 0 \wedge same-var\ xx\ (last\ cfs_3)\ (last\ cfs_4)$

⟨proof⟩

definition $\Delta 2' :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**
 $\Delta 2' = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $\quad (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $\quad statA$
 $\quad (cfg1, ibT1, ibUT1, ls1)$
 $\quad (cfg2, ibT2, ibUT2, ls2)$
 $\quad statO.$
 $(common\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $\quad (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $\quad statA$
 $\quad (cfg1, ibT1, ibUT1, ls1)$
 $\quad (cfg2, ibT2, ibUT2, ls2)$
 $\quad statO \wedge$
 $\quad same-var-o\ xx\ cfg3\ cfgs3\ cfg4\ cfgs4 \wedge$
 $\quad pcOf\ cfg3 = startOfIfThen \wedge$
 $\quad whileSpeculation\ (cfgs3!0)\ (last\ cfgs3) \wedge$
 $\quad misSpecL2\ cfgs3 \wedge misSpecL2\ cfgs4 \wedge$
 $\quad w1 = 2$
 $\quad))$

lemmas $\Delta 2'-defs = \Delta 2'-def\ common-def\ PC-def\ same-var-def$
 $\quad startOfElseBranch-def\ startOfIfThen-def$
 $\quad whileSpec-defs\ misSpecL2-def$

lemma $\Delta 2'-implies: \Delta 2' num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $\quad (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $\quad statA$
 $\quad (cfg1, ibT1, ibUT1, ls1)$
 $\quad (cfg2, ibT2, ibUT2, ls2)$
 $\quad statO \implies$
 $\quad pcOf\ cfg3 = 7 \wedge pcOf\ cfg4 = 7 \wedge$
 $\quad pcOf\ (last\ cfgs3) = pcOf\ (last\ cfgs4) \wedge$
 $\quad whileSpeculation\ (cfgs3!0)\ (last\ cfgs3) \wedge$
 $\quad whileSpeculation\ (cfgs4!0)\ (last\ cfgs4) \wedge$
 $\quad length\ cfgs3 = 2 \wedge length\ cfgs4 = 2$
 $\quad \langle proof \rangle$

definition $\Delta 3 :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**
 $\Delta 3 = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $\quad (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $\quad statA$
 $\quad (cfg1, ibT1, ibUT1, ls1)$
 $\quad (cfg2, ibT2, ibUT2, ls2)$
 $\quad statO.$

(common w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge
 same-var-o xx cfg3 cfs3 cfg4 cfs4 \wedge
 pcOf cfg3 = startOfElseBranch \wedge pcOf (last cfs3) \in inThenIfBeforeOutput \wedge
 misSpecL1 cfs3 \wedge
 (pcOf (last cfs3) = 7 \longrightarrow w1 = ∞) \wedge
 (pcOf (last cfs3) = 8 \longrightarrow w1 = 2) \wedge
 (pcOf (last cfs3) = 9 \longrightarrow w1 = 1)
))

lemmas $\Delta 3$ -defs = $\Delta 3$ -def common-def PC-def same-var-o-def
 startOfElseBranch-def inThenIfBeforeOutput-def

lemma $\Delta 3$ -implies: $\Delta 3$ num w1 w2 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 pcOf (last cfs3) \in inThenIfBeforeOutput \wedge
 pcOf (last cfs4) = pcOf (last cfs3) \wedge
 pcOf cfg3 = 12 \wedge pcOf cfg3 = pcOf cfg4 \wedge
 length cfs3 = Suc 0 \wedge length cfs4 = Suc 0
 <proof>

definition $\Delta e :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow$
 $stateV \Rightarrow status \Rightarrow bool$ **where**

$\Delta e = (\lambda num w1 w2 (pstate3, cfg3, cfs3, ib3, ls3)$
 (pstate4, cfg4, cfs4, ib4, ls4)
 statA
 (cfg1, ib1, ls1)
 (cfg2, ib2, ls2)
 statO.
 (pcOf cfg3 = endPC \wedge pcOf cfg4 = endPC \wedge cfs3 = [] \wedge cfs4 = [] \wedge
 pcOf cfg1 = endPC \wedge pcOf cfg2 = endPC))

lemmas Δe -defs = Δe -def common-def endPC-def

lemma *init*: *initCond* $\Delta 0$
 <proof>

lemma *step0: unwindIntoCond Δ_0 (oor Δ_0 Δ_1)*
<proof>

lemma *step1: unwindIntoCond Δ_1 (oor5 Δ_1 Δ_1' Δ_2 Δ_3 Δ_e)*
<proof>

lemma *step2: unwindIntoCond Δ_2 (oor3 Δ_2 Δ_2' Δ_1)*
<proof>

lemma *step3: unwindIntoCond Δ_3 (oor Δ_3 Δ_1)*
<proof>

lemma *step4: unwindIntoCond Δ_1' Δ_1*
<proof>

lemma *step5: unwindIntoCond Δ_2' Δ_2*
<proof>

lemma *stepe: unwindIntoCond Δ_e Δ_e*
<proof>

lemmas *theConds = step0 step1 step2 step3 step4 step5 stepe*

proposition *lrsecure*
<proof>

end

14 Proof of Relative Security for fun2

```

theory Fun-mask
  imports
    ../Instance-IMP/Instance-Secret-IMem
    Relative-Security.Unwinding-fin
begin

```

14.1 Function definition and Boilerplate

```

no-notation bot ( $\perp$ )

```

```

consts NN :: int
consts SS :: int

```

```

definition aa1 :: avname where aa1 = "a1"
definition aa2 :: avname where aa2 = "a2"
definition vv :: avname where vv = "v"
definition ii :: avname where ii = "i"
definition temp :: avname where temp = "temp"

```

```

lemmas vvars-defs = aa1-def aa2-def vv-def ii-def temp-def

```

```

lemma vvars-dff[simp]:
  aa1  $\neq$  aa2 aa1  $\neq$  vv aa1  $\neq$  ii aa1  $\neq$  temp
  aa2  $\neq$  aa1 aa2  $\neq$  vv aa2  $\neq$  ii aa2  $\neq$  temp
  vv  $\neq$  aa1 vv  $\neq$  aa2 vv  $\neq$  ii vv  $\neq$  temp
  ii  $\neq$  aa1 ii  $\neq$  aa2 ii  $\neq$  vv ii  $\neq$  temp
  temp  $\neq$  aa1 temp  $\neq$  aa2 temp  $\neq$  vv temp  $\neq$  ii
  <proof>

```

```

consts size-aa1 :: nat
consts size-aa2 :: nat

```

```

fun initVstore :: vstore  $\Rightarrow$  bool where
  initVstore vst = True

```

```

fun initAvstore :: avstore  $\Rightarrow$  bool where
  initAvstore (Avstore as) = (as aa1 = (0, size-aa1)  $\wedge$  as aa2 = (size-aa1, size-aa2))

```

```

definition prog  $\equiv$ 
  [
     $\emptyset$  Start ,
     $\not\#$  Input T ii ,
     $\not\#$  IfJump (Less (V ii) (N (int (size-aa1)))) 3 7 ,
     $\not\#$  vv ::= (Times (VA aa1 (V ii)) (N 512)) ,
     $\not\#$  M temp I (Less (V ii) (N (int (size-aa1)))) T VA aa2 (V vv) E (N 0),
  ]

```

```

// Output U (V temp) ,
// Jump 8 ,
// Output U (N 0)
]

```

lemma cases-8: $(i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee i = 6 \vee i = 7 \vee i = 8 \vee i > 8$
 $\langle proof \rangle$

lemma cases-branch: $(i::pcounter) < 3 \vee i = 3 \vee i = 4 \vee i = 5 \vee i > 5$
 $\langle proof \rangle$

lemma ii-aa1-cases: $vs\ ii < int\ size-aa1 \vee vs\ ii \geq int\ size-aa1$ $\langle proof \rangle$

lemma is-If-pcOf[simp]:
 $pcOf\ cfg < 8 \implies is-IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 2$
 $\langle proof \rangle$

lemma is-If-pc[simp]:
 $pc < 8 \implies is-IfJump\ (prog\ !\ pc) \longleftrightarrow pc = 2$
 $\langle proof \rangle$

lemma eq-Fence-pc[simp]:
 $pc < 8 \implies prog\ !\ pc \neq Fence$
 $\langle proof \rangle$

lemma not-isInput3[simp]: $\neg is-getInput\ (prog\ !\ 3)$
 $\langle proof \rangle$

lemma not-is-Output[simp]: $\neg is-Output\ (prog\ !\ 3)$
 $\langle proof \rangle$

```

consts mispred :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
consts resolve :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
consts update :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  predState
consts initPstate :: predState
fun istate :: state  $\Rightarrow$  bool where
istate s = (initAvstore (getAvstore s))

```

interpretation Prog-Mispred-Init where
prog = prog **and** initPstate = initPstate **and**

mispred = *mispred* **and** *resolve* = *resolve* **and** *update* = *update* **and**
istate = *istate*
 ⟨*proof*⟩

abbreviation

stepB-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** →*B* 55)
where *x* →*B* *y* == *stepB* *x* *y*

abbreviation

stepsB-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** →*B** 55)
where *x* →*B** *y* == *star* *stepB* *x* *y*

abbreviation

stepM-abbrev :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒
bool (**infix** →*M* 55)
where *x* →*M* *y* == *stepM* *x* *y*

abbreviation

stepN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val*
llist × *loc set* ⇒ *bool* (**infix** →*N* 55)
where *x* →*N* *y* == *stepN* *x* *y*

abbreviation

stepsN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val*
llist × *loc set* ⇒ *bool* (**infix** →*N** 55)
where *x* →*N** *y* == *star* *stepN* *x* *y*

abbreviation

stepS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** →*S* 55)
where *x* →*S* *y* == *stepS* *x* *y*

abbreviation

stepsS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** →*S** 55)
where *x* →*S** *y* == *star* *stepS* *x* *y*

lemma *endPC[simp]*: *endPC* = 8
 ⟨*proof*⟩

lemma *isInput1[simp]*: *prog* ! *Suc* 0 = *Input* *T* *ii*
 ⟨*proof*⟩

lemma *is-getTrustedInput-pcOf[simp]*: *pcOf* *cfg* < 8 ⇒ *is-getInput* (*prog*!(*pcOf*

$cfg)) \longleftrightarrow pcOf\ cfg = 1$
 $\langle proof \rangle$

lemma *is-Output-pcOf[simp]*: $pcOf\ cfg < 8 \implies is-Output\ (prog!(pcOf\ cfg)) \longleftrightarrow$
 $pcOf\ cfg = 5 \vee pcOf\ cfg = 7$
 $\langle proof \rangle$

lemma *is-Output1[simp]*: $is-Output\ (prog\ !\ 5)$
 $\langle proof \rangle$

lemma *is-Output2[simp]*: $is-Output\ (prog\ !\ 7)$
 $\langle proof \rangle$

lemma *isSecV-pcOf[simp]*:
 $isSecV\ (cfg, ibT, ibUT) \longleftrightarrow pcOf\ cfg = 0$
 $\langle proof \rangle$

lemma *isSecO-pcOf[simp]*:
 $isSecO\ (pstate, cfg, cfs, ib, ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
 $\langle proof \rangle$

lemma *getActV-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1\ then\ lhd\ ibT\ else\ \perp)$
 $\langle proof \rangle$

lemma *getObsV-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 5 \vee pcOf\ cfg = 7\ then$
 $\ (outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $\ else\ \perp$
 $)$
 $\langle proof \rangle$

lemma *getActO-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getActO\ (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1 \wedge cfs = []\ then\ lhd\ ibT\ else\ \perp)$
 $\langle proof \rangle$

lemma *getObsO-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getObsO\ (pstate, cfg, cfs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 5 \vee pcOf\ cfg = 7) \wedge cfs = []\ then$

$(outOf (prog!(pcOf\ cfg)) (stateOf\ cfg), ls)$
 $else\ \perp$
 $)$
 $\langle proof \rangle$

lemma $eqSec-pcOf[simp]$:
 $eqSec\ (cfg1, ibT1, ibUT1, ls1)\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
 $\langle proof \rangle$

lemma $nextB-pc0[simp]$:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc0[simp]$:
 $readLocs\ (Config\ 0\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc1[simp]$:
 $ibT \neq [] \implies nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(ii := lhd\ ibT)))\ avst\ h\ p), ltl\ ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc1[simp]$:
 $readLocs\ (Config\ 1\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc1'[simp]$:
 $ibT \neq [] \implies nextB\ (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(ii := lhd\ ibT)))\ avst\ h\ p), ltl\ ibT, ibUT)$
 $\langle proof \rangle$

lemma $readLocs-pc1'[simp]$:
 $readLocs\ (Config\ (Suc\ 0)\ s) = \{\}$
 $\langle proof \rangle$

lemma $nextB-pc2-then[simp]$:

$vs\ ii < int\ size-aa1 \implies$
 $nextB\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc2-else[simp]*:
 $vs\ ii \geq int\ size-aa1 \implies$
 $nextB\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 7\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc2*:
 $nextB\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ ii < int\ size-aa1\ then\ 3\ else\ 7)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,$
 $ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc2[simp]*:
 $readLocs\ (Config\ 2\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextM-pc2-then[simp]*:
 $vs\ ii \geq int\ size-aa1 \implies$
 $nextM\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextM-pc2-else[simp]*:
 $vs\ ii < int\ size-aa1 \implies$
 $nextM\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 7\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextM-pc2*:
 $nextM\ (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ ii < int\ size-aa1\ then\ 7\ else\ 3)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,$
 $ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc3[simp]*:
 $nextB\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $(let\ l = array-loc\ aa1\ (nat\ (vs\ ii))\ avst$
 $in\ (Config\ 4\ (State\ (Vstore\ (vs(vv := h\ l * 512)))\ avst\ (Heap\ h)\ p)),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc3[simp]*:
 $readLocs\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa1\ (nat\ (vs\ ii))\}$

$avst\}$
 $\langle proof \rangle$

lemma *nextB-pc4-True[simp]*:

$vs\ ii < int\ size-aa1 \implies$
 $nextB\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $(let\ l = array-loc\ aa2\ (nat\ (vs\ vv))\ avst$
 $in\ (Config\ 5\ (State\ (Vstore\ (vs(temp := h\ l)))\ avst\ (Heap\ h)\ p)),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *nextB-pc4-False[simp]*:

$vs\ ii \geq int\ size-aa1 \implies$
 $nextB\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $((Config\ 5\ (State\ (Vstore\ (vs(temp := 0)))\ avst\ (Heap\ h)\ p)),\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc4-True[simp]*:

$vs\ ii < int\ size-aa1 \implies$
 $readLocs\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa2\ (nat\ (vs\ vv))$
 $avst\}$
 $\langle proof \rangle$

lemma *readLocs-pc4-False[simp]*:

$vs\ ii \geq int\ size-aa1 \implies$
 $readLocs\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc5[simp]*:

$nextB\ (Config\ 5\ s,\ ibT,\ ibUT) =$
 $(Config\ 6\ s,\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc5[simp]*:

$readLocs\ (Config\ 5\ s) = \{\}$
 $\langle proof \rangle$

lemma *nextB-pc6[simp]*:

$nextB\ (Config\ 6\ s,\ ibT,\ ibUT) = (Config\ 8\ s,\ ibT,\ ibUT)$
 $\langle proof \rangle$

lemma *readLocs-pc6[simp]*:
 $readLocs (Config\ 6\ s) = \{\}$
 ⟨proof⟩

lemma *nextB-pc7[simp]*:
 $nextB (Config\ 7\ s, ibT, ibUT) = (Config\ 8\ s, ibT, ibUT)$
 ⟨proof⟩

lemma *readLocs-pc7[simp]*:
 $readLocs (Config\ 7\ s) = \{\}$
 ⟨proof⟩

lemma *nextB-stepB-pc*:
 $pc < 8 \implies (pc = 1 \longrightarrow ibT \neq []) \implies$
 $(Config\ pc\ s, ibT, ibUT) \rightarrow_B nextB (Config\ pc\ s, ibT, ibUT)$
 ⟨proof⟩

lemma *not-finalB*:
 $pc < 8 \implies (pc = 1 \longrightarrow ibT \neq []) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
 ⟨proof⟩

lemma *finalB-pc-iff'*:
 $pc < 8 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibT = [])$
 ⟨proof⟩

lemma *finalB-pc-iff*:
 $pc \leq 8 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibT = [] \vee pc = 8)$
 ⟨proof⟩

lemma *finalB-pcOf-iff[simp]*:
 $pcOf\ cfg \leq 8 \implies$
 $finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibT = [] \vee pcOf\ cfg = 8)$
 ⟨proof⟩

lemma *finalS-cond:pcOf cfg < 8* $\implies cfs = [] \implies (pcOf\ cfg = 1 \longrightarrow ibT \neq LNil)$
 $\implies \neg finalS (pstate, cfg, cfs, ibT, ibUT, ls)$
 ⟨proof⟩

lemma *not-is-getTrustedInput*[simp]: $cfg = \text{Config } 3 \text{ (State (Vstore vs) (Astore as) (Heap h) p)} \implies \neg \text{is-getInput (prog ! pcOf cfg)}$
 ⟨proof⟩

lemma *notOutput4*[simp]: $\neg \text{is-Output (prog ! 4)}$ ⟨proof⟩

lemma *notInput4*[simp]: $\neg \text{is-getInput (prog ! 4)}$ ⟨proof⟩

lemma *notInput5*[simp]: $\neg \text{is-getInput (prog ! 5)}$ ⟨proof⟩

lemma *finalS-cond-spec*:

$pcOf\ cf g < 8 \implies$
 $((pcOf\ (last\ cf gs) = 3 \vee pcOf\ (last\ cf gs) = 4 \vee pcOf\ (last\ cf gs) = 5) \wedge pcOf\ cf g = 7)$
 $\vee (pcOf\ (last\ cf gs) = 7 \wedge pcOf\ cf g = 3) \implies$
 $length\ cf gs = Suc\ 0 \implies$
 $\neg \text{finalS (pstate, cf g, cf gs, ibT, ibUT, ls)}$
 ⟨proof⟩

end

14.2 Proof

theory *Fun-mask-secure*

imports *Fun-mask*

begin

definition *PC* $\equiv \{0..8\}$

definition *beforeInput* $= \{0,1\}$

definition *afterInput* $= \{2..7\}$

definition *startOfThenBranch* $= 3$

definition *inThenBranchBeforeOutput* $= \{3,4,5\}$

definition *startOfElseBranch* $= 7$

definition *beforeAssign-vv* $= \{0..3\}$

definition *common* $:: \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status} \Rightarrow \text{bool}$

where

common $= (\lambda(\text{pstate3}, \text{cfg3}, \text{cf gs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$

$(\text{pstate4}, \text{cfg4}, \text{cf gs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$

statA

$(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$

$(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$

statO.

```

(pstate3 = pstate4 ∧
 cfg1 = cfg3 ∧ cfg2 = cfg4 ∧
 pcOf cfg3 = pcOf cfg4 ∧ map pcOf cfs3 = map pcOf cfs4 ∧
 pcOf cfg3 ∈ PC ∧ pcOf (set cfs3) ⊆ PC ∧
 ///
 array-base aa1 (getAvstore (stateOf cfg3)) = array-base aa1 (getAvstore (stateOf
 cfg4)) ∧
 (∀ cfg3' ∈ set cfs3. array-base aa1 (getAvstore (stateOf cfg3')) = array-base aa1
 (getAvstore (stateOf cfg3))) ∧
 (∀ cfg4' ∈ set cfs4. array-base aa1 (getAvstore (stateOf cfg4')) = array-base aa1
 (getAvstore (stateOf cfg4))) ∧
 array-base aa2 (getAvstore (stateOf cfg3)) = array-base aa2 (getAvstore (stateOf
 cfg4)) ∧
 (∀ cfg3' ∈ set cfs3. array-base aa2 (getAvstore (stateOf cfg3')) = array-base aa2
 (getAvstore (stateOf cfg3))) ∧
 (∀ cfg4' ∈ set cfs4. array-base aa2 (getAvstore (stateOf cfg4')) = array-base aa2
 (getAvstore (stateOf cfg4))) ∧
 ///
 (statA = Diff → statO = Diff)))

```

lemma *common-implies: common*

```

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO ⇒
pcOf cfg1 < 9 ∧ pcOf cfg2 = pcOf cfg1
⟨proof⟩

```

definition $\Delta 0 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

```

Δ0 = (λnum
 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (common (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO ∧
 ibT3 ≠ [] ∧ ibT4 ≠ [] ∧ ibT1 = ibT3 ∧ ibT2 = ibT4 ∧
 pcOf cfg3 ∈ beforeInput ∧

```

$ls1 = ls3 \wedge ls2 = ls4 \wedge$
 $noMisSpec\ cfgs3$
 $)$

lemmas $\Delta 0\text{-defs} = \Delta 0\text{-def}\ common\text{-def}\ PC\text{-def}\ beforeInput\text{-def}\ noMisSpec\text{-def}$

lemma $\Delta 0\text{-implies: } \Delta 0\ n$

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $(pcOf\ cfg3 = 1 \longrightarrow ibT3 \neq LNil) \wedge$
 $(pcOf\ cfg4 = 1 \longrightarrow ibT4 \neq LNil) \wedge$
 $pcOf\ cfg1 < 8 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $cfgs3 = [] \wedge pcOf\ cfg3 < 8 \wedge$
 $cfgs4 = [] \wedge pcOf\ cfg4 < 8$
 $\langle proof \rangle$

definition $\Delta 1 :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$
 $\Rightarrow bool$ **where**

$\Delta 1 = (\lambda num$
 $(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $same\text{-var}\text{-o}\ ii\ cfg3\ cfgs3\ cfg4\ cfgs4 \wedge$
 $pcOf\ cfg3 \in afterInput \wedge$
 $Dist\ ls1\ ls2\ ls3\ ls4 \wedge$
 $noMisSpec\ cfgs3$
 $)$

lemmas $\Delta 1\text{-defs} = \Delta 1\text{-def}\ common\text{-def}\ PC\text{-def}\ afterInput\text{-def}\ same\text{-var}\text{-o}\text{-def}\ noMis\text{-Spec}\text{-def}$

lemma $\Delta 1\text{-implies: } \Delta 1\ num$

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

```

statA
(cfg1,ibT1,ibUT1,ls1)
(cfg2,ibT2,ibUT2,ls2)
statO  $\implies$ 
pcOf cfg1 < 8  $\wedge$ 
cfs3 = []  $\wedge$  pcOf cfs3  $\neq$  1  $\wedge$  pcOf cfs3 < 8  $\wedge$ 
cfs4 = []  $\wedge$  pcOf cfs4  $\neq$  1  $\wedge$  pcOf cfs4 < 8
<proof>

```

definition $\Delta 2 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

```

 $\Delta 2 = (\lambda \text{num}$ 
  (pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
  (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)
  statO.
  (common (pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
    (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
    statA
    (cfg1,ibT1,ibUT1,ls1)
    (cfg2,ibT2,ibUT2,ls2)
    statO  $\wedge$ 
    same-var-o ii cfg3 cfs3 cfg4 cfs4  $\wedge$ 
    pcOf cfg3 = startOfThenBranch  $\wedge$  cfs3  $\neq$  []  $\wedge$ 
    pcOf (last cfs3) = startOfElseBranch  $\wedge$ 
    Dist ls1 ls2 ls3 ls4  $\wedge$ 
    misSpecL1 cfs3
  ))

```

lemmas $\Delta 2\text{-defs} = \Delta 2\text{-def}$ common-def PC-def same-var-o-def $\text{startOfThenBranch-def}$
 $\text{startOfElseBranch-def}$
 misSpecL1-def

lemma $\Delta 2\text{-implies}$: $\Delta 2$ n (pstate3,cfg3,cfs3,ibT3,ibUT3,ls3)
 (pstate4,cfg4,cfs4,ibT4,ibUT4,ls4)
 statA
 (cfg1,ibT1,ibUT1,ls1)
 (cfg2,ibT2,ibUT2,ls2)
 statO \implies
 pcOf (last cfs3) = 7 \wedge pcOf cfg3 = 3 \wedge
 pcOf (last cfs4) = pcOf (last cfs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 length cfs3 = Suc 0 \wedge
 length cfs3 = length cfs4
 <proof>

definition $\Delta 3 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

$\Delta 3 = (\lambda \text{num}$
 (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (common (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge
 same-var-o ii cfg3 cfs3 cfg4 cfs4 \wedge
 vstore (getVstore (stateOf cfg3)) ii \geq int size-aa1 \wedge
 pcOf cfg3 = startOfElseBranch \wedge
 pcOf (last cfs3) \in inThenBranchBeforeOutput \wedge
 Dist ls1 ls2 ls3 ls4 \wedge
 misSpecL1 cfs3
))

lemma $\Delta 3\text{-def}' : \Delta 3 \text{ num}$

(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO =
 (common (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge
 same-var-o ii cfg3 cfs3 cfg4 cfs4 \wedge
 vstore (getVstore (stateOf cfg3)) ii \geq int size-aa1 \wedge
 pcOf cfg3 = startOfElseBranch \wedge
 pcOf (last cfs3) \in inThenBranchBeforeOutput \wedge
 Dist ls1 ls2 ls3 ls4 \wedge
 misSpecL1 cfs3
) \langle proof \rangle

lemmas $\Delta 3\text{-defs} = \Delta 3\text{-def}$ *common-def* *PC-def* *same-var-o-def*
startOfElseBranch-def *inThenBranchBeforeOutput-def*
beforeAssign-vv-def *misSpecL1-def*

lemma Δ_3 -implies: Δ_3 n (pstate3, cfg3, cfigs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \implies
 (pcOf (last cfigs3) = 3 \vee pcOf (last cfigs3) = 4 \vee pcOf (last cfigs3) = 5) \wedge
 pcOf cfg3 = 7 \wedge
 pcOf (last cfigs4) = pcOf (last cfigs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 array-base aa1 (getAvstore (stateOf (last cfigs3))) = array-base aa1 (getAvstore
 (stateOf cfg3)) \wedge
 array-base aa1 (getAvstore (stateOf (last cfigs4))) = array-base aa1 (getAvstore
 (stateOf cfg4)) \wedge
 length cfigs3 = Suc 0 \wedge
 length cfigs3 = length cfigs4
 <proof>

definition Δe :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status
 \Rightarrow bool **where**

Δe = (λ num
 (pstate3, cfg3, cfigs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO.
 (pcOf cfg3 = endPC \wedge pcOf cfg4 = endPC \wedge cfigs3 = [] \wedge cfigs4 = [] \wedge
 pcOf cfg1 = endPC \wedge pcOf cfg2 = endPC))

lemmas Δe -defs = Δe -def common-def endPC

lemma *init*: *initCond* Δ_0
 <proof>

lemma *step0*: *unwindIntoCond* Δ_0 (*oor* Δ_0 Δ_1)
 <proof>

lemma *step1*: *unwindIntoCond* Δ_1 (*oor* Δ_1 Δ_2 Δ_3 Δe)
 <proof>

lemma *step2: unwindIntoCond Δ_2 Δ_1*
<proof>

lemma *step3: unwindIntoCond Δ_3 (oor Δ_3 Δ_1)*
<proof>

lemma *stepe: unwindIntoCond Δ_e Δ_e*
<proof>

lemmas *theConds = step0 step1 step2 step3 stepe*
find-theorems *unwindIntoCond name: rsecure*

proposition *rsecure*
<proof>
end

[3] [1]

References

- [1] K. Cheang, C. Rasmussen, S. A. Seshia, and P. Subramanyan, “A formal approach to secure speculation,” in *CSF*. IEEE, 2019, pp. 288–303. [Online]. Available: <https://doi.org/10.1109/CSF.2019.00027>
- [2] B. Dongol, M. Griffin, A. Popescu, and J. Wright, “Relative security: Formally modeling and (dis)proving resilience against semantic optimization vulnerabilities,” in *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2024, pp. 409–424. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/CSF61375.2024.00027>
- [3] T. Nipkow and G. Klein, *Concrete Semantics: With Isabelle/HOL*. Springer, 2014.