

A formalized programming language with speculative execution

Jamie Wright Andrei Popescu

June 15, 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	17
3.1	State Transitions	17
3.2	Elimination Rules	18
4	Misprediction and Speculative Semantics	19
4.1	Misprediction Oracle	19
4.2	Mispredicting Step	20

4.2.1	State Transitions	20
4.3	Speculative Semantics	22
4.3.1	State Transitions	24
4.3.2	Elimination Rules	29
5	Relative Security instantiation - Common Aspects	30
6	Relative Security Instance: Secret Memory	35
7	Relative Security Instance: Secret Memory Input	39
8	Disproof of Relative Security for fun1	43
8.1	Function definition and Boilerplate	43
8.2	Proof	53
8.2.1	Concrete leak	53
8.2.2	Auxillary lemmas for disproof	58
8.2.3	Disproof of fun1	59
9	Proof of Relative Security for fun2	64
9.1	Function definition and Boilerplate	64
9.2	Proof	76
10	Proof of Relative Security for fun3	98
10.1	Function definition and Boilerplate	98
10.2	Proof	110
11	Proof of Relative Security for fun4	138
11.1	Function definition and Boilerplate	139
11.2	Proof	152
12	Proof of Relative Security for fun5	200
12.1	Function definition and Boilerplate	201
12.2	Proof	219
13	Proof of Relative Security for fun6	268
13.1	Function definition and Boilerplate	268
13.2	Proof	288
14	Proof of Relative Security for fun2	339
14.1	Function definition and Boilerplate	339
14.2	Proof	353

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*
by (*simp add: array-loc-def*)

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* ⇒ *aexp* ⇒ *bexp* **where**
Eq a1 a2 ≡ *And (Not (Less a1 a2)) (Not (Less a2 a1))*

lemma *Eq-verif*: *bval (Eq a1 a2) s* ⇔ *aval a1 s = aval a2 s*
apply *standard*
unfolding *Eq-def* **by** *simp+*

fun *outOf* :: *com* ⇒ *state* ⇒ *val* **where**
outOf c s = (*case c of Output T aexp* ⇒ *aval aexp s* | - ⇒ *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-prog: *prog* ≠ [] ∧ *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 $lenth-prog-gt-0$: $length\ prog > 0$
using $wf-prog$ **by** $auto$

lemma $lenth-prog-not-0$: $length\ prog \neq 0$
using $wf-prog$ **by** $auto$

lemma $endPC-gt-0$: $endPC > 0$
unfolding $endPC-def$ **using** $lenth-prog-gt-0$ **by** $blast$

lemma $endPC-not-0$: $endPC \neq 0$
unfolding $endPC-def$ **using** $lenth-prog-not-0$ **by** $blast$

lemma $hd-prog-Start$: $hd\ prog = Start$
using $wf-prog$ **by** $auto$

lemma $prog-0$: $prog ! 0 = Start$
by ($metis\ hd-conv-nth\ wf-prog$)

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:

$$\begin{aligned}
& pc < endPC \implies prog!pc = (arr[index] ::= a) \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 \\
& (Config\ pc\ s,\ ibT,\ ibUT) \\
& \rightarrow B \\
& (Config\ (Suc\ pc)\ (State\ vst\ avst\ (Heap\ (h(l := w)))\ p),\ ibT,\ ibUT) \\
& | \\
& \text{getTrustedInput:} \\
& pc < endPC \implies prog!pc = Input\ T\ x \implies \\
& (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ LCons\ i\ ibT,\ ibUT) \\
& \rightarrow B \\
& (Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := i)))\ avst\ h\ p),\ ibT,\ ibUT) \\
& | \\
& \text{getUntrustedInput:} \\
& pc < endPC \implies prog!pc = Input\ U\ x \implies \\
& (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ LCons\ i\ ibUT) \\
& \rightarrow B \\
& (Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := i)))\ avst\ h\ p),\ ibT,\ ibUT) \\
& | \\
& \text{Output:} \\
& pc < endPC \implies prog!pc = Output\ t\ aexp \implies \\
& (Config\ pc\ s,\ ibT,\ ibUT) \\
& \rightarrow B \\
& (Config\ (Suc\ pc)\ s,\ ibT,\ ibUT) \\
& | \\
& \text{Jump:} \\
& pc < endPC \implies prog!pc = Jump\ pc1 \implies \\
& (Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc1\ s,\ ibT,\ ibUT) \\
& | \\
& \text{IfTrue:} \\
& pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies \\
& bval\ b\ s \implies \\
& (Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc1\ s,\ ibT,\ ibUT) \\
& | \\
& \text{IfFalse:} \\
& pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies \\
& \neg\ bval\ b\ s \implies \\
& (Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc2\ s,\ ibT,\ ibUT) \\
& | \\
& \text{MaskTrue:} \\
& 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 \\
& (Config\ pc\ s,\ ibT,\ ibUT) \\
& \rightarrow B \\
& (Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := aval\ a1\ s)))\ avst\ h\ p),\ ibT,\ ibUT)
\end{aligned}$$

|
MaskFalse:
 $pc < endPC \implies prog!pc = (M\ x\ I\ b\ T\ a1\ E\ a2) \implies$
 $\neg bval\ b\ s \implies$
 $s = State\ (Vstore\ vs)\ avst\ h\ p \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT)$
 $\rightarrow B$
 $(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := aval\ a2\ s))))\ avst\ h\ p),\ ibT,\ ibUT)$

lemmas *stepB-induct* = *stepB.induct*[*split-format*(*complete*)]

abbreviation

stepsB :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒ *bool* (**infix**
 $\rightarrow B^*$ 55)
where $x \rightarrow B^* y == star\ stepB\ x\ y$

declare *stepB.intros*[*simp,intro*]

2.3 State Transitions

Useful lemmas regarding valid transitions of the semantics along with conditions for termination (*finalB*)

definition *finalB* = *final stepB*

lemmas *finalB-defs* = *final-def finalB-def*

lemma *finalB-iff-aux:*

$pc < endPC \wedge$
 $(\forall x\ i\ a.\ prog!pc = (x[i] ::= a) \longrightarrow aval\ i\ s \geq 0 \wedge$
 $aval\ i\ s < int\ (array-bound\ x\ (getAvstore\ s))) \wedge$
 $(\forall y.\ prog!pc = Input\ T\ y \longrightarrow ibT \neq LNil) \wedge$
 $(\forall y.\ prog!pc = Input\ U\ y \longrightarrow ibUT \neq LNil)$
 \longleftrightarrow
 $(\exists\ cfg'.\ (Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ cfg')$

apply (*cases s*) **subgoal for** *vst avst h p*

apply(*cases vst*) **apply**(*cases h*) **subgoal for** *vs hh* **apply** *clarsimp*

apply (*cases prog!pc*)

subgoal by (*auto elim: stepB.cases, blast*)

subgoal by (*auto elim: stepB.cases, blast*)

subgoal for *t* **apply**(*cases t*)

subgoal by(*cases ibT, auto elim: stepB.cases, blast*)

subgoal by(*cases ibUT, auto elim: stepB.cases, blast*) .

subgoal for *t* **apply**(*cases t*)

subgoal by (*auto elim: stepB.cases, blast*)

subgoal by (*auto elim: stepB.cases, blast*) .

subgoal by (*auto elim: stepB.cases, blast*)

subgoal by (*auto elim: stepB.cases, blast*)
subgoal by (*auto elim: stepB.cases, blast*)
subgoal by (*auto elim: stepB.cases, blast*)
subgoal by (*auto elim: stepB.cases, blast*)
subgoal by (*auto elim: stepB.cases, blast*) . . .

lemma *finalB-iff*:

finalB (*Config pc s, ibT, ibUT*)

\longleftrightarrow

(*pc* \geq *endPC* \vee

($\exists x\ i\ a.$ *prog!pc* = (*x*[*i*] ::= *a*) \wedge

(\neg *aval i s* \geq 0 \vee \neg *aval i s* < *int (array-bound x (getAstore s))*)) \vee

($\exists y.$ *prog!pc* = *Input T y* \wedge *ibT* = *LNil*) \vee

($\exists y.$ *prog!pc* = *Input U y* \wedge *ibUT* = *LNil*)

using *finalB-iff-aux*[*of pc s ibT ibUT*] **unfolding** *finalB-def final-def*

using *verit-comp-simplify1* (3) **by** *blast*

lemma *stepB-determ*:

cfg-ib \rightarrow_B *cfg-ib'* \implies *cfg-ib* \rightarrow_B *cfg-ib''* \implies *cfg-ib''* = *cfg-ib'*

apply(*induction arbitrary: cfg-ib'' rule: stepB.induct*)

by (*auto elim: stepB.cases*)

definition *nextB* :: *config* \times *val llist* \times *val llist* \Rightarrow *config* \times *val llist* \times *val llist*

where

nextB *cfg-ib* \equiv *SOME* *cfg'-ib'*. *cfg-ib* \rightarrow_B *cfg'-ib'*

lemma *nextB-stepB*: \neg *finalB* *cfg-ib* \implies *cfg-ib* \rightarrow_B (*nextB* *cfg-ib*)

unfolding *nextB-def* **apply**(*rule someI-ex*)

unfolding *finalB-def final-def* **by** *auto*

lemma *stepB-nextB*: *cfg-ib* \rightarrow_B *cfg'-ib'* \implies *cfg'-ib'* = *nextB* *cfg-ib*

unfolding *nextB-def* **apply**(*rule sym*) **apply**(*rule some-equality*)

using *stepB-determ* **by** *auto*

lemma *nextB-iff-stepB*: \neg *finalB* *cfg-ib* \implies *nextB* *cfg-ib* = *cfg'-ib'* \longleftrightarrow *cfg-ib* \rightarrow_B *cfg'-ib'*

using *nextB-stepB stepB-nextB* **by** *blast*

lemma *stepB-iff-nextB*: *cfg-ib* \rightarrow_B *cfg'-ib'* \longleftrightarrow \neg *finalB* *cfg-ib* \wedge *nextB* *cfg-ib* = *cfg'-ib'*

by (*metis finalB-def final-def stepB-nextB*)

2.3.1 Simplification Rules

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

lemma *nextB-Start-Skip-Fence[simp]*:
 $pc < endPC \implies prog!pc \in \{Start, Skip, Fence\} \implies$
 $nextB (Config\ pc\ s, ibT, ibUT) = (Config (Suc\ pc)\ s, ibT, ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

lemma *nextB-Assign[simp]*:
 $pc < endPC \implies prog!pc = (x ::= a) \implies$
 $s = State (Vstore\ vs)\ avst\ h\ p \implies$
 $nextB (Config\ pc\ s, ibT, ibUT)$
 $=$
 $(Config (Suc\ pc) (State (Vstore (vs(x := aval\ a\ s))))\ avst\ h\ p,$
 $ibT, ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

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
 $nextB (Config\ pc\ s, ibT, ibUT)$
 $=$
 $(Config (Suc\ pc) (State\ vst\ avst (Heap (h(l := w)))\ p), ibT, ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

lemma *nextB-getTrustedInput[simp]*:
 $pc < endPC \implies prog!pc = (Input\ T\ x) \implies$
 $nextB (Config\ pc (State (Vstore\ vs)\ avst\ h\ p), LCons\ i\ ibT, ibUT)$
 $=$
 $(Config (Suc\ pc) (State (Vstore (vs(x := i)))\ avst\ h\ p), ibT, ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

lemma *nextB-getUntrustedInput[simp]*:
 $pc < endPC \implies prog!pc = (Input\ U\ x) \implies$
 $nextB (Config\ pc (State (Vstore\ vs)\ avst\ h\ p), ibT, LCons\ i\ ibUT)$
 $=$
 $(Config (Suc\ pc) (State (Vstore (vs(x := i)))\ avst\ h\ p), ibT, ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

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

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

lemma *nextB-Output[simp]*:
 $pc < endPC \implies prog!pc = Output\ t\ aexp \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
 $=$
 $(Config\ (Suc\ pc)\ s,\ ibT,\ ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

lemma *nextB-Jump[simp]*:
 $pc < endPC \implies prog!pc = Jump\ pc1 \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc1\ s,\ ibT,\ ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*, *simp-all+*)

lemma *nextB-IfTrue[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $bval\ b\ s \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc1\ s,\ ibT,\ ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

lemma *nextB-IfFalse[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $\neg\ bval\ b\ s \implies$
 $nextB\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc2\ s,\ ibT,\ ibUT)$
by(intro *stepB-nextB[THEN sym]* *stepB.intros*)

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

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*)
by(*intro stepB-nextB*[*THEN sym*] *stepB.intros*)

lemma *finalB-endPC*: *pcOf cfg = endPC* \implies *finalB* (*cfg*, *ibT*, *ibUT*)
by (*metis finalB-iff config.collapse le-eq-less-or-eq*)

lemma *stepB-endPC*: *pcOf cfg = endPC* \implies \neg (*cfg*, *ibT*, *ibUT*) $\rightarrow B$ (*cfg'*, *ibT'*, *ibUT'*)
by (*simp add: stepB-iff-nextB finalB-endPC*)

lemma *stepB-imp-le-endPC*: **assumes** (*cfg*, *ibT*, *ibUT*) $\rightarrow B$ (*cfg'*, *ibT'*, *ibUT'*)
shows *pcOf cfg < endPC*
using *assms* **by**(*cases rule: stepB.cases, simp-all*)

lemma *stepB-0*: (*Config 0 s*, *ibT*, *ibUT*) $\rightarrow B$ (*Config 1 s*, *ibT*, *ibUT*)
using *prog-0* **by** (*simp add: endPC-gt-0*)

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 (*cfg*, *ibT*, *ibUT*) $\rightarrow B$ (*cfg'*, *ibT'*, *ibUT'*) \rangle
and \langle *cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*)) \rangle
and \langle *cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*)) \rangle
and \langle *prog!pc* \in {*Start*, *Skip*, *Fence*} \rangle
shows \langle *vs'* = *vs* \wedge *ibT* = *ibT'* \wedge *ibUT* = *ibUT'* \wedge
pc' = *Suc pc* \wedge *avst'* = *avst* \wedge *h'* = *h* \wedge
p' = *p* \rangle
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

lemma *stepB-AssignE*:
assumes \langle (*cfg*, *ibT*, *ibUT*) $\rightarrow B$ (*cfg'*, *ibT'*, *ibUT'*) \rangle
and \langle *cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*)) \rangle
and \langle *cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*)) \rangle
and \langle *prog!pc* = (*x ::= a*) \rangle
shows \langle *vs'* = (*vs*(*x := aval a* (*stateOf cfg*))) \wedge
ibT = *ibT'* \wedge *ibUT* = *ibUT'* \wedge *pc'* = *Suc pc* \wedge
avst' = *avst* \wedge *h'* = *h* \wedge *p'* = *p* \rangle
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

lemma *stepB-getTrustedInputE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = Input\ T\ x \rangle$
shows $\langle vs' = (vs(x := lhd\ ibT)) \wedge$
 $ibT' = ltl\ ibT \wedge ibUT = ibUT' \wedge pc' = Suc\ pc \wedge$
 $avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* $(cfg, ibT, ibUT)$ $(cfg', ibT', ibUT')$ *rule: stepB.cases*)
by *auto*

lemma *stepB-getUntrustedInputE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = Input\ U\ x \rangle$
shows $\langle vs' = (vs(x := lhd\ ibUT)) \wedge$
 $ibT' = ibT \wedge ibUT' = ltl\ ibUT \wedge pc' = Suc\ pc \wedge$
 $avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* $(cfg, ibT, ibUT)$ $(cfg', ibT', ibUT')$ *rule: stepB.cases*)
by *auto*

lemma *stepB-OutputE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = Output\ t\ aexp \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = Suc\ pc \wedge avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* $(cfg, ibT, ibUT)$ $(cfg', ibT', ibUT')$ *rule: stepB.cases*)
by *auto*

lemma *stepB-JumpE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = Jump\ pc1 \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = pc1 \wedge avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* $(cfg, ibT, ibUT)$ $(cfg', ibT', ibUT')$ *rule: stepB.cases*)
by *auto*

lemma *stepB-IfTrueE*:

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

$pc' = pc1 \wedge avst' = avst \wedge h' = h \wedge p' = p$

using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

lemma *stepB-IfFalseE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = IfJump\ b\ pc1\ pc2 \rangle$ **and** $\langle \neg bval\ b\ (stateOf\ cfg) \rangle$
shows $\langle vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
 $pc' = pc2 \wedge avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

lemma *stepB-MaskTrueE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = M\ x\ I\ b\ T\ a1\ E\ a2 \rangle$ **and** $\langle bval\ b\ (stateOf\ cfg) \rangle$
shows $\langle vs' = (vs(x := aval\ a1\ (stateOf\ cfg))) \wedge$
 $ibT = ibT' \wedge ibUT = ibUT' \wedge pc' = Suc\ pc \wedge$
 $avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

lemma *stepB-MaskFalseE*:

assumes $\langle (cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT') \rangle$
and $\langle cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)) \rangle$
and $\langle cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')) \rangle$
and $\langle prog!pc = M\ x\ I\ b\ T\ a1\ E\ a2 \rangle$ **and** $\langle \neg bval\ b\ (stateOf\ cfg) \rangle$
shows $\langle vs' = (vs(x := aval\ a2\ (stateOf\ cfg))) \wedge$
 $ibT = ibT' \wedge ibUT = ibUT' \wedge pc' = Suc\ pc \wedge$
 $avst' = avst \wedge h' = h \wedge p' = p \rangle$
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*, *ibUT'*) *rule: stepB.cases*)
by *auto*

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* \Rightarrow *state* \Rightarrow *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** $\langle \rightarrow N \rangle$ 55)

where

$(cfg, ibT, ibUT, ls) \rightarrow N (cfg', ibT', ibUT', ls') =$
 $((cfg, ibT, ibUT) \rightarrow B (cfg', ibT', ibUT')) \wedge ls' = ls \cup readLocs\ cfg)$

abbreviation

stepsN :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *loc set* ⇒ *bool* (**infix** $\langle \rightarrow N^* \rangle$ 55)

where $x \rightarrow N^* y == star\ stepN\ x\ y$

definition *finalN* = *final stepN*

lemmas *finalN-defs* = *final-def finalN-def*

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

$finalN\ (cfg, ibT, ibUT, ls) \longleftrightarrow finalB\ (cfg, ibT, ibUT)$

unfolding *finalN-def finalB-def final-def* **by** *auto*

3.1 State Transitions

fun *nextN* :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *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*: $\neg finalN\ cfg\ ib\ ls \implies cfg\ ib\ ls \rightarrow N\ (nextN\ cfg\ ib\ ls)$

apply(cases *cfg-ib-ls*)
using *Prog.stepB-nextB Prog-axioms finalN-def*
final-def nextN.simps old.prod.case stepN.elims(2)
by *force*

lemma *stepN-nextN*: $cfg\text{-}ib\text{-}ls \rightarrow N\ cfg'\text{-}ib'\text{-}ls' \implies cfg'\text{-}ib'\text{-}ls' = nextN\ cfg\text{-}ib\text{-}ls$
apply(cases *cfg-ib-ls*) **apply**(cases *cfg'-ib'-ls'*)
using *Prog.stepB-nextB Prog-axioms* **by** *auto*

lemma *nextN-iff-stepN*:
 $\neg\ finalN\ cfg\text{-}ib\text{-}ls \implies nextN\ cfg\text{-}ib\text{-}ls = cfg'\text{-}ib'\text{-}ls' \iff cfg\text{-}ib\text{-}ls \rightarrow N\ cfg'\text{-}ib'\text{-}ls'$
using *nextN-stepN stepN-nextN* **by** *blast*

lemma *stepN-iff-nextN*: $cfg\text{-}ib\text{-}ls \rightarrow N\ cfg'\text{-}ib'\text{-}ls' \iff \neg\ finalN\ cfg\text{-}ib\text{-}ls \wedge nextN\ cfg\text{-}ib\text{-}ls = cfg'\text{-}ib'\text{-}ls'$
by (*metis finalN-def final-def stepN-nextN*)

lemma *finalN-endPC*: $pcOf\ cfg = endPC \implies finalN\ (cfg, ibT, ibUT)$
by (*metis finalN-iff-finalB finalB-endPC old.prod.exhaust*)

lemma *stepN-endPC*: $pcOf\ cfg = endPC \implies \neg\ (cfg, ibT, ibUT) \rightarrow N\ (cfg', ibT', ibUT')$
by (*simp add: finalN-endPC stepN-iff-nextN*)

lemma *stepN-0*: $(Config\ 0\ s, ibT, ibUT, ls) \rightarrow N\ (Config\ 1\ s, ibT, ibUT, ls)$
using *prog-0 One-nat-def stepB-0* **by** (*auto simp: readLocs-def*)

lemma *finalB-eq-finalN*: $finalB\ (cfg, ibT, ibUT) \iff (\forall\ ls.\ finalN\ (cfg, ibT, ibUT, ls))$
unfolding *finalN-defs finalB-def*
apply *standard* **by** *auto*

3.2 Elimination Rules

lemma *stepN-Assign2E*:
assumes $\langle\ cfg1, ibT1, ibUT1, ls1 \rangle \rightarrow N\ \langle\ cfg1', ibT1', ibUT1', ls1' \rangle$
and $\langle\ cfg2, ibT2, ibUT2, ls2 \rangle \rightarrow N\ \langle\ cfg2', ibT2', ibUT2', ls2' \rangle$
and $\langle\ cfg1 = (Config\ pc1\ (State\ (Vstore\ vs1)\ avst1\ h1\ p1)) \rangle$ **and** $\langle\ cfg1' = (Config\ pc1'\ (State\ (Vstore\ vs1')\ avst1'\ h1'\ p1')) \rangle$
and $\langle\ cfg2 = (Config\ pc2\ (State\ (Vstore\ vs2)\ avst2\ h2\ p2)) \rangle$ **and** $\langle\ cfg2' = (Config\ pc2'\ (State\ (Vstore\ vs2')\ avst2'\ h2'\ p2')) \rangle$
and $\langle\ prog!pc1 = (x ::= a) \rangle$ **and** $\langle\ pcOf\ cfg1 = pcOf\ cfg2 \rangle$
shows $\langle\ vs1' = (vs1\ (x :=\ aval\ a\ (stateOf\ cfg1))) \wedge ibT1 = ibT1' \wedge ibUT1 = ibUT1' \wedge$
 $vs2' = (vs2\ (x :=\ aval\ a\ (stateOf\ cfg2))) \wedge ibT2 = ibT2' \wedge ibUT2 = ibUT2' \wedge$
 $pc1' = Suc\ pc1 \wedge pc2' = Suc\ pc2 \wedge ls2' = ls2 \cup readLocs\ cfg2 \wedge$

```

      avst1' = avst1 ∧ avst2' = avst2 ∧ ls1' = ls1 ∪ readLocs cfg1
using assms apply clarsimp
apply (drule stepB-AssignE[of - - - - - pc1 vs1 avst1 h1 p1
      pc1' vs1' avst1' h1' p1' x a], clarify+)
apply (drule stepB-AssignE[of - - - - - pc2 vs2 avst2 h2 p2
      pc2' vs2' avst2' h2' p2' x a], clarify+)
by auto

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⟩
using assms apply clarsimp
apply (drule stepB-Seq-Start-Skip-FenceE[of - - - - - pc1 vs1 avst1 h1 p1
      pc1' vs1' avst1' h1' p1'], clarify+)
apply (drule stepB-Seq-Start-Skip-FenceE[of - - - - - pc2 vs2 avst2 h2 p2
      pc2' vs2' avst2' h2' p2'], clarify+)
by (auto simp add: readLocs-def)

```

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  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
and resolve :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  bool
and update :: predState  $\Rightarrow$  pcounter list  $\Rightarrow$  predState
begin

```

4.2 Mispredicting Step

stepM simply goes the other way than stepB at branches

```

inductive
stepM :: config  $\times$  val llist  $\times$  val llist  $\Rightarrow$  config  $\times$  val llist  $\times$  val llist  $\Rightarrow$  bool (infix
 $\rightarrow M$  55)
where
IfTrue[intro]:
pc < endPC  $\Longrightarrow$  prog!pc = IfJump b pc1 pc2  $\Longrightarrow$ 
  bval b s  $\Longrightarrow$ 
  (Config pc s, ibT, ibUT)  $\rightarrow M$  (Config pc2 s, ibT, ibUT)
|
IfFalse[intro]:
pc < endPC  $\Longrightarrow$  prog!pc = IfJump b pc1 pc2  $\Longrightarrow$ 
   $\neg$  bval b s  $\Longrightarrow$ 
  (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')
apply (cases s) subgoal for vst avst h p apply clarsimp

```

```

apply (cases prog!pc)
  subgoal by (auto elim: stepM.cases)
  subgoal by (auto elim: stepM.cases)
  subgoal by (auto elim: stepM.cases)
  subgoal by (auto elim: stepM.cases)
  subgoal by (auto elim: stepM.cases)
  subgoal by (auto elim: stepM.cases)

```

subgoal by (*auto elim: stepM.cases*)
subgoal by (*auto elim: stepM.cases*)
subgoal by (*auto elim: stepM.cases*)
subgoal by (*auto elim: stepM.cases,meson IfFalse IfTrue*) . .

lemma *finalM-iff*:
 $finalM (Config\ pc\ (State\ vst\ avst\ h\ p),\ ibT,\ ibUT)$
 \longleftrightarrow
 $(pc \geq endPC \vee \neg is\text{-}IfJump\ (prog!pc))$
using *finalM-iff-aux* **unfolding** *finalM-def final-def*
by (*metis linorder-not-less*)

lemma *finalB-imp-finalM*:
 $finalB (cfg,\ ibT,\ ibUT) \implies finalM (cfg,\ ibT,\ ibUT)$
apply(*cases cfg*) **subgoal for** *pc s* **apply**(*cases s*)
subgoal for *vst avst h p* **apply** *clarsimp* **unfolding** *finalB-iff finalM-iff* **by** *auto*
. .

lemma *not-finalM-imp-not-finalB*:
 $\neg finalM (cfg,\ ibT,\ ibUT) \implies \neg finalB (cfg,\ ibT,\ ibUT)$
using *finalB-imp-finalM* **by** *blast*

lemma *stepM-determ*:
 $cfg\text{-}ib \rightarrow M\ cfg\text{-}ib' \implies cfg\text{-}ib \rightarrow M\ cfg\text{-}ib'' \implies cfg\text{-}ib'' = cfg\text{-}ib'$
apply(*induction arbitrary: cfg-ib'' rule: stepM.induct*)
by (*auto elim: stepM.cases*)

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

lemma *nextM-stepM*: $\neg finalM\ cfg\text{-}ib \implies cfg\text{-}ib \rightarrow M\ (nextM\ cfg\text{-}ib)$
unfolding *nextM-def* **apply**(*rule someI-ex*)
unfolding *finalM-def final-def* **by** *auto*

lemma *stepM-nextM*: $cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib' \implies cfg'\text{-}ib' = nextM\ cfg\text{-}ib$
unfolding *nextM-def* **apply**(*rule sym*) **apply**(*rule some-equality*)
using *stepM-determ* **by** *auto*

lemma *nextM-iff-stepM*: $\neg finalM\ cfg\text{-}ib \implies nextM\ cfg\text{-}ib = cfg'\text{-}ib' \longleftrightarrow cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib'$
using *nextM-stepM stepM-nextM* **by** *blast*

lemma *stepM-iff-nextM*: $cfg\text{-}ib \rightarrow M\ cfg'\text{-}ib' \longleftrightarrow \neg finalM\ cfg\text{-}ib \wedge nextM\ cfg\text{-}ib = cfg'\text{-}ib'$
by (*metis finalM-def final-def stepM-nextM*)

lemma *nextM-IfTrue[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $\neg bval\ b\ s \implies$
 $nextM\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc1\ s,\ ibT,\ ibUT)$
by(intro *stepM-nextM[THEN sym]* *stepM.intros*)

lemma *nextM-IfFalse[simp]*:
 $pc < endPC \implies prog!pc = IfJump\ b\ pc1\ pc2 \implies$
 $bval\ b\ s \implies$
 $nextM\ (Config\ pc\ s,\ ibT,\ ibUT) = (Config\ pc2\ s,\ ibT,\ ibUT)$
by(intro *stepM-nextM[THEN sym]* *stepM.intros*)

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 speculaton 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* × *config* × *config list* × *val llist* × *val llist* × *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

$stepS :: configS \Rightarrow configS \Rightarrow bool$ (**infix** $\rightarrow S$ 55)

where

nonspec-normal:

$cfgs = [] \implies$

$\neg isIfJump (prog!(pcOf\ cfg)) \vee \neg mispred\ pstate\ [pcOf\ cfg] \implies$

$pstate' = pstate \implies$

$\neg finalB\ (cfg,\ ibT,\ ibUT) \implies (cfg',\ ibT',\ ibUT') = nextB\ (cfg,\ ibT,\ ibUT) \implies$

$cfgs' = [] \implies$

$ls' = ls \cup readLocs\ cfg$

\implies

$(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \rightarrow S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$

|

nonspec-mispred:

$cfgs = [] \implies$

$isIfJump\ (prog!(pcOf\ cfg)) \implies mispred\ pstate\ [pcOf\ cfg] \implies$

$pstate' = update\ pstate\ [pcOf\ cfg] \implies$

$\neg finalM\ (cfg,\ ibT,\ ibUT) \implies (cfg',\ ibT',\ ibUT') = nextB\ (cfg,\ ibT,\ ibUT) \implies$

$(cfg1',\ ibT1',\ ibUT1') = nextM\ (cfg,\ ibT,\ ibUT) \implies$

$cfgs' = [cfg1'] \implies$

$ls' = ls \cup readLocs\ cfg$

\implies

$(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \rightarrow S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$

|

spec-normal:

$cfgs \neq [] \implies$

$\neg resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs) \implies$

$\neg isIfJump\ (prog!(pcOf\ (last\ cfgs))) \vee \neg mispred\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs) \implies$

$prog!(pcOf\ (last\ cfgs)) \neq Fence \implies$

$pstate' = pstate \implies$

$\neg is-getInput\ (prog!(pcOf\ (last\ cfgs))) \implies$

$\neg is-Output\ (prog!(pcOf\ (last\ cfgs))) \implies$

$\neg finalB\ (last\ cfgs,\ ibT,\ ibUT) \implies (cfg1',\ ibT',\ ibUT') = nextB\ (last\ cfgs,\ ibT,\ ibUT) \implies$

$$\begin{aligned}
& \text{cfg}' = \text{cfg} \implies \text{cfgs}' = \text{butlast } \text{cfgs} @ [\text{cfg1}] \implies \\
& \text{ls}' = \text{ls} \cup \text{readLocs } (\text{last } \text{cfgs}) \\
& \implies \\
& (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) \rightarrow_S (\text{pstate}', \text{cfg}', \text{cfgs}', \text{ibT}', \text{ibUT}', \text{ls}') \\
& | \\
& \text{spec-mispred:} \\
& \text{cfgs} \neq [] \implies \\
& \neg \text{resolve } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \implies \\
& \text{is-IfJump } (\text{prog}!(\text{pcOf } (\text{last } \text{cfgs}))) \implies \text{mispred } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \\
& \implies \\
& \text{pstate}' = \text{update } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \implies \\
& \neg \text{finalM } (\text{last } \text{cfgs}, \text{ibT}, \text{ibUT}) \implies \\
& (\text{lcfg}', \text{ibt}', \text{ibUT}') = \text{nextB } (\text{last } \text{cfgs}, \text{ibT}, \text{ibUT}) \implies (\text{cfg1}', \text{ibt1}', \text{ibUT1}') = \\
& \text{nextM } (\text{last } \text{cfgs}, \text{ibt}, \text{ibUT}) \implies \\
& \text{cfg}' = \text{cfg} \implies \text{cfgs}' = \text{butlast } \text{cfgs} @ [\text{lcfg}'] @ [\text{cfg1}'] \implies \\
& \text{ls}' = \text{ls} \cup \text{readLocs } (\text{last } \text{cfgs}) \\
& \implies \\
& (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) \rightarrow_S (\text{pstate}', \text{cfg}', \text{cfgs}', \text{ibT}', \text{ibUT}', \text{ls}') \\
& | \\
& \text{spec-Fence:} \\
& \text{cfgs} \neq [] \implies \\
& \neg \text{resolve } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \implies \\
& \text{prog}!(\text{pcOf } (\text{last } \text{cfgs})) = \text{Fence} \implies \\
& \text{pstate}' = \text{pstate} \implies \text{cfg}' = \text{cfg} \implies \text{cfgs}' = [] \implies \\
& \text{ibt} = \text{ibt}' \implies \text{ibUT} = \text{ibUT}' \implies \text{ls}' = \text{ls} \\
& \implies \\
& (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) \rightarrow_S (\text{pstate}', \text{cfg}', \text{cfgs}', \text{ibt}', \text{ibUT}', \text{ls}') \\
& | \\
& \text{spec-resolve:} \\
& \text{cfgs} \neq [] \implies \\
& \text{resolve } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \vee \text{is-Output } (\text{prog}!(\text{pcOf } (\text{last } \text{cfgs}))) \vee \\
& \text{is-getInput } (\text{prog}!(\text{pcOf } (\text{last } \text{cfgs}))) \implies \\
& \text{pstate}' = \text{update } \text{pstate} (\text{pcOf } \text{cfg} \# \text{map } \text{pcOf } \text{cfgs}) \implies \\
& \text{cfg}' = \text{cfg} \implies \text{cfgs}' = \text{butlast } \text{cfgs} \implies \\
& \text{ibt} = \text{ibt}' \implies \text{ibUT} = \text{ibUT}' \implies \text{ls}' = \text{ls} \\
& \implies \\
& (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) \rightarrow_S (\text{pstate}', \text{cfg}', \text{cfgs}', \text{ibt}', \text{ibUT}', \text{ls}')
\end{aligned}$$

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

4.3.1 State Transitions

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

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

$cfgs' = [] \wedge ls' = ls \cup readLocs\ cfgs$
apply(subst stepS.simps) **by** auto

lemma stepS-nonspec-normal-iff1[simp]:
 $cfgs = [] \implies \neg is\text{-IfJump}\ (prog!\ pc) \vee \neg mispred\ pstate\ [pc]$
 \implies
 $(pstate, (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)),\ cfgs,\ ibT,\ ibUT,\ ls) \rightarrow_S\ (pstate',$
 $(Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')), cfgs', ibT', ibUT', ls')$
 \iff
 $(pstate' = pstate \wedge \neg finalB\ ((Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)),\ ibT,\ ibUT)$
 \wedge
 $((Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p')), ibT', ibUT') = nextB\ ((Config\ pc$
 $(State\ (Vstore\ vs)\ avst\ h\ p)),\ ibT,\ ibUT) \wedge$
 $cfgs' = [] \wedge ls' = ls \cup readLocs\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p)))$
using stepS-nonspec-normal-iff config.sel(1) **by** presburger

lemma stepS-nonspec-mispred-iff[simp]:
 $cfgs = [] \implies is\text{-IfJump}\ (prog!(pcOf\ cfg)) \implies mispred\ pstate\ [pcOf\ cfg]$
 \implies
 $(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \rightarrow_S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$
 \iff
 $(\exists\ cfg1'\ ibT1'\ ibUT1'.\ pstate' = update\ pstate\ [pcOf\ cfg] \wedge$
 $\neg finalM\ (cfg,\ ibT,\ ibUT) \wedge (cfg',\ ibT',\ ibUT') = nextB\ (cfg,\ ibT,\ ibUT) \wedge$
 $(cfg1',\ ibT1',\ ibUT1') = nextM\ (cfg,\ ibT,\ ibUT) \wedge$
 $cfgs' = [cfg1'] \wedge ls' = ls \cup readLocs\ cfg)$
apply(subst stepS.simps) **by** auto

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
 $(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \rightarrow_S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$
 \iff
 $(\exists\ cfg1'.\ pstate' = pstate \wedge$
 $\neg finalB\ (last\ cfgs,\ ibT,\ ibUT) \wedge (cfg1',\ ibT',\ ibUT') = nextB\ (last\ cfgs,\ ibT,$
 $ibUT) \wedge$
 $cfg' = cfg \wedge cfgs' = butlast\ cfgs\ @\ [cfg1'] \wedge ls' = ls \cup readLocs\ (last\ cfgs))$
apply(subst stepS.simps) **by** auto

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
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \iff
 $(\exists cfg1' ibT1' ibUT1' lcfg'. pstate' = update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs) \wedge$
 $\neg\ finalM\ (last\ cfgs, ibT, ibUT) \wedge$
 $(lcfg', ibT', ibUT') = nextB\ (last\ cfgs, ibT, ibUT) \wedge$
 $(cfg1', ibT1', ibUT1') = nextM\ (last\ cfgs, ibT, ibUT) \wedge$
 $cfg' = cfg \wedge cfgs' = butlast\ cfgs\ @\ [lcfg']\ @\ [cfg1'] \wedge ls' = ls \cup readLocs\ (last$
 $cfgs))$
apply(subst stepS.simps) **by auto**

lemma stepS-spec-Fence-iff[simp]:
 $cfgs \neq [] \implies$
 $\neg\ resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs) \implies$
 $prog!(pcOf\ (last\ cfgs)) = Fence$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \iff
 $(pstate' = pstate \wedge cfg = cfg' \wedge cfgs' = [] \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' =$
 $ls)$
apply(subst stepS.simps) **by auto**

lemma stepS-spec-resolve-iff[simp]:
 $cfgs \neq [] \implies$
 $resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs)$
 \implies
 $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow_S (pstate', cfg', cfgs', ibT', ibUT', ls')$
 \iff
 $(pstate' = update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfgs) \wedge$
 $cfg' = cfg \wedge cfgs' = butlast\ cfgs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' = ls)$
apply(subst stepS.simps) **by auto**

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

$cfgs = []$
 $\neg\ isIfJump\ (prog!(pcOf\ cfg)) \vee \neg\ mispred\ pstate\ [pcOf\ cfg]$
 $pstate' = pstate$
 $\neg\ finalB\ (cfg, ibT, ibUT)$
 $(cfg', ibT', ibUT') = nextB\ (cfg, ibT, ibUT)$
 $cfgs' = []$

$ls' = ls \cup \text{readLocs } cfg$
|

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

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

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

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

|

```
cfgs ≠ []
  resolve pstate (pcOf cfg # map pcOf cfgs)
  pstate' = update pstate (pcOf cfg # map pcOf cfgs)
  cfg' = cfg
  cfgs' = butlast cfgs
  ls' = ls
  ibT' = ibT
  ibUT' = ibUT
```

|

```
cfgs ≠ []
  is-getInput (prog!(pcOf (last cfgs)))
  pstate' = update pstate (pcOf cfg # map pcOf cfgs)
  cfg' = cfg
  cfgs' = butlast cfgs
  ls' = ls
  ibT' = ibT
  ibUT' = ibUT
```

|

```
cfgs ≠ []
  is-Output (prog!(pcOf (last cfgs)))
  pstate' = update pstate (pcOf cfg # map pcOf cfgs)
  cfg' = cfg
  cfgs' = butlast cfgs
  ls' = ls
  ibT' = ibT
  ibUT' = ibUT
using assms by (cases rule: stepS.cases,metis+)
```

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

```
apply(cases ss')
apply safe apply(cases rule: stepS-cases, auto)
  using finalB-endPC apply blast
  using finalB-endPC apply blast
  using finalB-endPC finalB-imp-finalM by blast
```

abbreviation

```
stepsS :: configS ⇒ configS ⇒ bool (infix →S* 55)
where x →S* y ≡ star stepS x y
```

definition *finalS* = final stepS

lemmas *finalS-defs* = final-def finalS-def

lemma *stepS-determ*:

```
cfg-ib →S cfg-ib' ⇒ cfg-ib →S cfg-ib'' ⇒ cfg-ib'' = cfg-ib'
```

```

apply(induction arbitrary: cfg-ib'' rule: stepS.induct)
subgoal for cfgs cfg pstate pstate' ibT ibUT
  by(cases nextB (cfg, ibT, ibUT), auto elim: stepS.cases)
subgoal for cfgs cfg pstate pstate' ibT ibUT
  apply(cases nextB (cfg, ibT, ibUT), cases nextM (cfg, ibT, ibUT))
  by(auto elim: stepS.cases)
subgoal for cfgs pstate cfg pstate' ibT ibUT
  by(cases nextB (last cfgs, ibT, ibUT), auto elim: stepS.cases)
subgoal for cfgs cfg pstate pstate' ibT ibUT
  apply(cases nextB (last cfgs, ibT, ibUT), cases nextM (last cfgs, ibT, ibUT))
  by(auto elim: stepS.cases)
subgoal by(auto elim: stepS.cases)
subgoal by(auto elim: stepS.cases) .

```

```

lemma stepS-0: (pstate, Config 0 s, [], ibT, ibUT, ls) →S (pstate, Config 1 s, [],
ibT, ibUT, ls)
using prog-0 apply—apply(rule nonspec-normal)
using One-nat-def stebB-0 stepB-nextB
by (auto simp: readLocs-def finalB-def final-def, meson)

```

```

lemma stepS-imp-stepB:(pstate, cfg, [], ibT,ibUT, ls) →S (pstate', cfg', cfs',
ibT',ibUT', ls') ⇒ (cfg, ibT,ibUT) →B (cfg', ibT',ibUT')
subgoal premises s
  using s apply (cases rule: stepS-cases)
  by (metis finalB-imp-finalM stepB-iff-nextB)+ .

```

4.3.2 Elimination Rules

```

lemma stepS-Assign2E:
  assumes  $\langle ps3, cfg3, cfs3, ibT3, ibUT3, ls3 \rangle \rightarrow S (ps3', cfg3', cfs3', ibT3', ibUT3',$ 
 $ls3') \rangle$ 
  and  $\langle ps4, cfg4, cfs4, ibT4, ibUT4, ls4 \rangle \rightarrow S (ps4', cfg4', cfs4', ibT4', ibUT4',$ 
 $ls4') \rangle$ 
  and  $\langle cfg3 = (Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)) \rangle$  and  $\langle cfg3' =$ 
 $(Config\ pc3'\ (State\ (Vstore\ vs3')\ avst3'\ h3'\ p3')) \rangle$ 
  and  $\langle cfg4 = (Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)) \rangle$  and  $\langle cfg4' =$ 
 $(Config\ pc4'\ (State\ (Vstore\ vs4')\ avst4'\ h4'\ p4')) \rangle$ 
  and  $\langle cfs3 = [] \rangle$  and  $\langle cfs4 = [] \rangle$ 
  and  $\langle prog!pc3 = (x ::= a) \rangle$  and  $\langle pcOf\ cfg3 = pcOf\ cfg4 \rangle$ 
shows  $\langle cfs3' = [] \wedge cfs4' = [] \wedge$ 
 $vs3' = (vs3(x := aval\ a\ (stateOf\ cfg3))) \wedge$ 
 $vs4' = (vs4(x := aval\ a\ (stateOf\ cfg4))) \wedge$ 
 $pc3' = Suc\ pc3 \wedge pc4' = Suc\ pc4 \wedge ls4' = ls4 \cup readLocs\ cfg4 \wedge$ 
 $avst3' = avst3 \wedge avst4' = avst4 \wedge ls3' = ls3 \cup readLocs\ cfg3 \wedge$ 
 $p3 = p3' \wedge p4 = p4' \rangle$ 
using assms apply clarify
apply—apply(frule stepS-imp-stepB[of ps3])
apply(frule stepS-imp-stepB[of ps4])
apply (drule stepB-AssignE[of - - - - - pc3 vs3 avst3 h3 p3)

```

```

      pc3' vs3' avst3' h3' p3' x a], clarify+)
apply (drule stepB-AssignE[of - - - - - pc4' vs4' avst4' h4' p4'
      pc4' vs4' avst4' h4' p4'], clarify+)
by fastforce+

```

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 its (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 (cfgs::config list)  $\equiv$  (cfgs = [])
lemma noMisSpec-ext[simp]:map x cfgs = map x cfgs'  $\implies$  noMisSpec cfgs  $\longleftrightarrow$ 
noMisSpec cfgs'
by (auto simp: noMisSpec-def)

```

```

definition misSpecL1 where misSpecL1 (cfgs::config list)  $\equiv$  (length cfgs = Suc 0)

```

```

lemma misSpecL1-len:misSpecL1 cfgs  $\longleftrightarrow$  length cfgs = 1 by (simp add: mis-
SpecL1-def)

```

```

lemma misSpecL1-ne:misSpecL1 cfgs  $\implies$  cfgs  $\neq$  [] by (auto simp add: mis-
SpecL1-def)

```

```

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

```

```

fun tuple::'a × 'b × 'c ⇒ 'a × 'b
  where tuple (a,b,c) = (a,b)

fun tuple-sel::'a × 'b × 'c × 'd × 'e ⇒ 'b × 'd
  where tuple-sel (a,b,c,d,e) = (b,d)

fun cfgsOf::'a × 'b × 'c × 'd × 'e ⇒ 'c
  where cfgsOf (a,b,c,d,e) = c

fun pstateOf::'a × 'b × 'c × 'd × 'e ⇒ 'a
  where pstateOf (a,b,c,d,e) = a

fun stateOfs::'a × 'b × 'c × 'd × 'e ⇒ '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 × val llist × val llist × loc set
fun validTransV where validTransV (cfg-ib-ls, cfg-ib-ls') = cfg-ib-ls →N cfg-ib-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 × 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)
by (simp add: stepN-iff-nextN)+

```

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 ⇒ pcounter list ⇒ bool
and resolve :: predState ⇒ pcounter list ⇒ bool
and update :: predState ⇒ pcounter list ⇒ 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 $istateV\text{-}config\text{-}imp:$
 $istateV (cfg, ibT, ibUT, ls) \Longrightarrow pcOf\ cfg = 0 \wedge ls = \{\} \wedge ibT \neq LNil$
by *force*

lemma $istateO\text{-}config\text{-}imp:$
 $istateO (pstate, cfg, cfs, ibT, ibUT, ls) \Longrightarrow$
 $cfs = [] \wedge pcOf\ cfg = 0 \wedge ls = \{\} \wedge ibT \neq LNil$
unfolding $istateO.simps$
by *auto*

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

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

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

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
 $\forall\ cfg3' \in set\ (butlast\ cfs).$
 $vstore\ (getVstore\ (stateOf\ cfg3'))\ var =$
 $vstore\ (getVstore\ (stateOf\ cfg))\ var$
by (*meson in-set-butlastD*)

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

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

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
 $array-base\ aa1\ (getAvstore\ (stateOf\ cfg)) =$
 $array-base\ aa1\ (getAvstore\ (stateOf\ (last\ cfs)))$
by *simp*

lemma *finalB-imp-finalS*: $finalB\ (cfg,\ ibT,\ ibUT) \implies (\forall\ pstate\ cfs\ ls.\ finalS\ (pstate,\ cfg,\ [],\ ibT,\ ibUT,\ ls))$
unfolding *finalB-def finalS-def final-def* **apply** *clarsimp*
subgoal for $pstate\ ls\ pstate'\ cfg'\ cfs'\ ibT\ ibUT'\ ls'$
apply (*erule alle[of - (cfg', ibT, ibUT')]*)
subgoal premises *step*
using *step(1)* **apply** (*cases rule: stepS-cases*)

using *finalB-imp-finalM step(2) nextB-stepB* **by** (*simp-all, blast*) . .

lemma *cfgs-Suc-zero*[*simp*]: $\text{length } \text{cfgs} = \text{Suc } 0 \implies \text{cfgs} = [\text{last } \text{cfgs}]$
by (*metis Suc-length-conv last-ConsL length-0-conv*)

lemma *cfgs-map*[*simp*]: $\text{length } \text{cfgs} = \text{Suc } 0 \implies \text{map } \text{pcOf } \text{cfgs} = [\text{pcOf } (\text{last } \text{cfgs})]$
apply (*frule cfgs-Suc-zero*[*of cfgs*])
apply (*rule ssubst*[*of map pcOf cfgs map pcOf [last cfgs]*])
by (*presburger, metis list.simps(8,9)*)

lemma *is-misSpecL1*[*simp*]: *misSpecL1* [a] **unfolding** *misSpecL1-def* **by** *auto*

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 *cfgs* = [])

```

fun getSecO :: stateO ⇒ secret where
  getSecO (pstate, cfg, cfgs, ibT, ibUT, ls) = stateOf cfg
lemma isSecV-iff: isSecV ss ⟷ pcOf (fst ss) = 0
  unfolding isSecV-def
  by (simp add: case-prod-beta)

lemma validTransO-iff-nextS: validTransO (s1, s2) = (¬ finalS s1 ∧ (stepS s1
s2))
  unfolding finalS-def final-def
by simp (metis old.prod.exhaust)

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
  apply standard
  subgoal by (simp add: finalN-defs)
  subgoal for s by (cases s, simp)
  subgoal for s apply(cases s) subgoal for cfg ibT ibUT ls apply(cases cfg)
subgoal for n st
  unfolding isSecV-def
  using stebB-0[of st ibT ibUT] stepB-iff-nextB by fastforce . .
  subgoal by (simp add: finalS-defs)
  subgoal by (simp add: finalS-defs)
  subgoal for ss apply(cases ss) subgoal for ps cfg cfgs ibT ibUT ls apply(cases
cfg) subgoal for n st
  unfolding isSecO-def finalS-def final-def
  using stepS-0[of ps st ibT ibUT ls] by auto . . .

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]*:
 $(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ } s \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhs } \text{ibT}$
by (cases prog!(pcOf cfg), auto simp: Van.getAct-def)

lemma *not-is-getTrustedInput-getActV[simp]*:
 $\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{noninform}$
apply (cases prog!(pcOf cfg), auto simp: Van.getAct-def)
subgoal for x **by** (cases x , simp-all) .

lemma *is-Output-getObsV[simp]*:
 $(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Output } U \text{ } \text{out} \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
by (cases prog!(pcOf cfg), auto simp: Van.getObs-def)

lemma *not-is-Output-getObsV[simp]*:
 $\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
apply (cases prog!(pcOf cfg), auto simp: Van.getObs-def)
subgoal for x **by** (cases x , simp-all) .

lemma *is-getTrustedInput-Nil-getActO[simp]*:
 $(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ } s \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhs } \text{ibT}$
by (cases prog!(pcOf cfg), auto simp: Opt.getAct-def)

lemma *not-is-getTrustedInput-Nil-getActO[simp]*:
 $\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg}))$
 $\vee \text{cfgs} \neq [] \implies \text{getActO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
apply (cases cfgs, auto)
apply (cases prog!(pcOf cfg), auto simp: Opt.getAct-def)
subgoal for x **by** (cases x , simp-all) .

lemma *is-Output-Nil-getObsO[simp]*:
 $\text{prog!}(\text{pcOf } \text{cfg}) = \text{Output } U \text{ } s \implies$
 $\text{getObsO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = (\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
by (cases prog!(pcOf cfg), auto simp: Opt.getObs-def)

lemma *not-is-Output-Nil-getObsO[simp]*:
 $\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \vee \text{cfgs} \neq [] \implies \text{getObsO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls})$
 $= \perp$
apply (cases cfgs, auto)
apply (cases prog!(pcOf cfg), auto simp: Opt.getObs-def)
subgoal for x **by** (cases x , simp-all) .

lemma *getActV-simps*:
 $\text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{case } \text{prog!}(\text{pcOf } \text{cfg}) \text{ of}$
 $\quad \text{Input } T \text{ } - \Rightarrow \text{lhs } \text{ibT}$
 $\quad | \text{Input } U \text{ } - \Rightarrow \text{lhs } \text{ibUT}$

```

    |-  $\Rightarrow \perp$ 
  )
  unfolding Van.getAct-def
  apply (simp split: com.splits, safe)
  subgoal for t by(cases t, simp-all)
  subgoal for t by(cases t, simp-all) .

lemma getObsV-simps:
getObsV (cfg,ibT,ibUT,ls) =
  (case prog!(pcOf cfg) of
    Output U -  $\Rightarrow$  (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
    |-  $\Rightarrow \perp$ 
  )
unfolding Van.getObs-def
apply (simp split: com.splits, safe)
subgoal for t by(cases t, simp-all)
subgoal for t by(cases t, simp-all) .

lemma getActO-simps:
getActO (pstate,cfg,cfgs,ibT,ibUT,ls) =
  (case (cfgs,prog!(pcOf cfg)) of
    ( $[],Input T -$ )  $\Rightarrow$  lhd ibT
    |( $[],Input U -$ )  $\Rightarrow$  lhd ibUT
    |-  $\Rightarrow \perp$ 
  )
apply (simp split: com.splits list.splits, safe)
unfolding Opt.getAct-def
subgoal for t by(cases t, simp-all) .

lemma getObsO-simps:
getObsO (pstate,cfg,cfgs,ibT,ibUT,ls) =
  (case (cfgs,prog!(pcOf cfg)) of
    ( $[],Output U -$ )  $\Rightarrow$  (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
    |-  $\Rightarrow \perp$ 
  )
unfolding Opt.getObs-def
apply (simp split: com.splits list.splits, safe)
subgoal for t by(cases t, simp-all)
subgoal for t by(cases t, simp-all) .

```

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* (\perp)

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]:¬is-IfJump writeSecretOnFile* **by** (*simp add:*)

lemma *writeOnFile-not-Inp[simp]:¬is-getInput writeSecretOnFile* **by** (*simp add:*)

lemma *writeOnFile-not-Fence[simp]:writeSecretOnFile ≠ Fence* **by** (*simp add:*)

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

lemma *ffileVal-vstore[simp]:ffileVal* *cfg* = *vstoreOf(cfg) ffile* **by**(*simp add: ffile-Val-def*)

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* ⇒ *bool* **where**

isSecV *ss* ≡ *case ss of (cfg,ibT,ibUT,ls) ⇒ ¬finalN ss*

fun *getSecV* :: *stateV* ⇒ *secret* **where**

getSecV (*cfg,ibT,ibUT,ls*) =
(*case prog!(pcOf* *cfg)* *of*
 Start ⇒ (*stateOf* *cfg*, \perp , \perp)
 | *Input T* - ⇒ (\perp , *lhd* *ibT*, \perp)
 | - ⇒ (\perp , \perp , \perp))

lemma *isSecV-iff:isSecV ss* \longleftrightarrow \neg *finalN ss*
unfolding *isSecV-def*
by (*simp add: case-prod-beta*)

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-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*
apply *standard*
subgoal **by** (*simp add: finalN-defs*)
subgoal **for** *s* **by** (*cases s, simp*)
subgoal **by** (*simp add: isSecV-def*)
subgoal **by** (*simp add: finalS-defs*)
subgoal **by** (*simp add: finalS-defs*)
subgoal **for** *ss* **apply**(*cases ss*) **subgoal** **for** *ps cfg cfgs ib ls* **apply**(*cases cfg*)
subgoal **for** *n s*
unfolding *isSecO-def finalS-def final-def*
using *stepS-0[of ps s ib ls]* **by** *auto . . .*

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-getInputT-getActV*[*simp*]:

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } U \text{ inp} \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibUT}$
by (*cases prog!(pcOf cfg)*, *auto simp: Van.getAct-def*)

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

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ inp} \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibT}$
by (*cases prog!(pcOf cfg)*, *auto simp: Van.getAct-def*)

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

$\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getActV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
apply (*cases prog!(pcOf cfg)*, *auto simp: Van.getAct-def*)
subgoal for t apply(*cases t, simp-all*) . .

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

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Output } U \text{ out} \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) =$
 $(\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
by (*cases prog!(pcOf cfg)*, *auto simp: Van.getObs-def*)

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

$\neg \text{is-Output } (\text{prog!}(\text{pcOf } \text{cfg})) \implies \text{getObsV } (\text{cfg}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
apply (*cases prog!(pcOf cfg)*, *auto simp: Van.getObs-def*)
subgoal for t apply(*cases t, simp-all*) . .

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

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } T \text{ inp} \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibT}$
by (*cases prog!(pcOf cfg)*, *auto simp: Opt.getAct-def*)

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

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Input } U \text{ inp} \implies \text{getActO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = \text{lhd } \text{ibUT}$
by (*cases prog!(pcOf cfg)*, *auto simp: Opt.getAct-def*)

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

$(\neg \text{is-getInput } (\text{prog!}(\text{pcOf } \text{cfg})))$
 $\vee \text{cfgs} \neq [] \implies \text{getActO } (\text{pstate}, \text{cfg}, \text{cfgs}, \text{ibT}, \text{ibUT}, \text{ls}) = \perp$
apply (*cases cfgs, auto*)
apply (*cases prog!(pcOf cfg)*, *auto simp: Opt.getAct-def*)
subgoal for t apply(*cases t, simp-all*) . .

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

$(\text{prog!}(\text{pcOf } \text{cfg})) = \text{Output } U \text{ out} \implies$
 $\text{getObsO } (\text{pstate}, \text{cfg}, [], \text{ibT}, \text{ibUT}, \text{ls}) = (\text{outOf } (\text{prog!}(\text{pcOf } \text{cfg})) (\text{stateOf } \text{cfg}), \text{ls})$
by (*cases prog!(pcOf cfg)*, *auto simp: Opt.getObs-def*)

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

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

apply (*cases cfgs, auto*)
apply (*cases prog!(pcOf cfg), auto simp: Opt.getObs-def*)
subgoal for t apply(*cases t, simp-all*) . .

lemma *getActV-simps*:
 $getActV (cfg, ibT, ibUT, ls) =$
 (*case prog!(pcOf cfg) of*
 Input T - \Rightarrow lhd ibT
 | *Input U - \Rightarrow lhd ibUT*
 | - $\Rightarrow \perp$
)
unfolding *Van.getAct-def*
apply (*simp split: com.splits, safe*)
subgoal for t apply(*cases t, simp-all*) .
subgoal for t apply(*cases t, simp-all*) . .

lemma *getObsV-simps*:
 $getObsV (cfg, ibT, ibUT, ls) =$
 (*case prog!(pcOf cfg) of*
 Output U - \Rightarrow (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
 | - $\Rightarrow \perp$
)
unfolding *Van.getObs-def*
apply (*simp split: com.splits, safe*)
subgoal for t apply(*cases t, simp-all*) .
subgoal for t apply(*cases t, simp-all*) . .

lemma *getActO-simps*:
 $getActO (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 (*case (cfgs, prog!(pcOf cfg)) of*
 ($[], Input T -$) \Rightarrow *lhd ibT*
 | ($[], Input U -$) \Rightarrow *lhd ibUT*
 | - $\Rightarrow \perp$
)
unfolding *Van.getAct-def*
apply (*simp split: com.splits list.splits, safe*)
subgoal for t apply(*cases t, simp-all*) . .

lemma *getObsO-simps*:
 $getObsO (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 (*case (cfgs, prog!(pcOf cfg)) of*
 ($[], Output U -$) \Rightarrow (*outOf (prog!(pcOf cfg)) (stateOf cfg), ls*)
 | - $\Rightarrow \perp$
)
unfolding *Opt.getObs-def*

```

apply (simp split: com.splits list.splits, safe)
subgoal for t apply(cases t, simp-all) .
subgoal for t apply(cases t, simp-all) . .

```

end

end

8 Disproof of Relative Security for fun1

```

theory Fun1
imports ../Instance-IMP/Instance-Secret-IMem
  Secret-Directed-Unwinding.SD-Unwinding-fin
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 = "tt"

```

```

lemma NN-suc[simp]:nat (NN + 1) = Suc (nat NN)
  by force

```

```

lemma NN:NN ≥ 0 by auto

```

```

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
unfolding vvars-defs by auto

```

```

consts size-aa1 :: nat
consts size-aa2 :: nat

```

definition $s\text{-add} = \{a. a \neq \text{nat } NN+1\}$

fun $vs_0 :: \text{char list} \Rightarrow \text{int}$ **where**

$vs_0 \ x = 0$

lemma $vs_0[simp]: (\lambda x. 0) = vs_0$ **unfolding** $vs_0.simps$ **by** $simp$

fun $as :: \text{char list} \Rightarrow \text{nat} \times \text{nat}$ **where**

$as \ a = (\text{if } a = aa1 \text{ then } (0, \text{nat } NN)$
 $\quad \text{else } (\text{if } a = aa2 \text{ then } (\text{nat } NN, \text{nat size-aa2})$
 $\quad \text{else } (\text{nat size-aa2}, 0))$

definition $avst' \equiv (\text{Avstore } as)$

lemmas $avst\text{-defs} = avst'\text{-def } as.simps$

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

$avst'$

unfolding $avst\text{-defs}$ **by** $auto$

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

fun $initVstore :: \text{vstore} \Rightarrow \text{bool}$ **where**

$initVstore \ (Vstore \ vst) = (vst = vs_0)$

fun $initAvstore :: \text{avstore} \Rightarrow \text{bool}$ **where**

$initAvstore \ avst = (avst = avst')$

fun $initHeap :: (\text{nat} \Rightarrow \text{int}) \Rightarrow \text{bool}$ **where**

$initHeap \ h = (\forall x \in read\text{-add}. h \ x = 0)$

lemma $initAvstore\text{-0}[intro]: initAvstore \ avst' \Longrightarrow \text{array-base } aa1 \ avst' = 0$

unfolding $avst\text{-defs}$ array-base-def

by $(smt \ (verit, del\text{-insts}) \ avstore.case \ fstI)$

fun $istate :: \text{state} \Rightarrow \text{bool}$ **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* $(Less \ (V \ xx) \ (N \ NN)) \ 4 \ 5$,

$\not\#$ $tt ::= (VA \ aa2 \ (Times \ (VA \ aa1 \ (V \ xx)) \ (N \ 512)))$,

```

  ! 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
apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
. . . . .

```

```

lemma xx-NN-cases: vs xx < (int NN)  $\vee$  vs xx  $\geq$  (int NN) by auto

```

```

lemma is-If-pcOf[simp]:
pcOf cfg < 6  $\implies$  is-IfJump (prog ! (pcOf cfg))  $\longleftrightarrow$  pcOf cfg = 3
apply(cases cfg) subgoal for pc s using cases-5[of pcOf cfg]
apply (auto simp: prog-def) . .

```

```

lemma is-If-pc[simp]:
pc < 6  $\implies$  is-IfJump (prog ! pc)  $\longleftrightarrow$  pc = 3
using cases-5[of pc]
by (auto simp: prog-def)

```

```

lemma eq-Fence-pc[simp]:
pc < 6  $\implies$  prog ! pc  $\neq$  Fence
using cases-5[of pc]
by (auto simp: prog-def)

```

```

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*
by (*standard*, *simp add: prog-def*)

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* = 6
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getTrustedInput-pcOf[simp]*: *pcOf* *cfg* < 6 ⇒ *is-getInput* (*prog!*(*pcOf*
cfg)) ⇔ *pcOf* *cfg* = 1
using *cases-5[of pcOf cfg]* **by** (*auto simp: prog-def*)

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

lemma *is-Output-pcOf*[simp]: $\text{pcOf } cfg < 6 \implies \text{is-Output } (\text{prog!}(\text{pcOf } cfg)) \longleftrightarrow$
 $\text{pcOf } cfg = 5 \vee \text{pcOf } cfg = 6$
using *cases-5*[of *pcOf cfg*] **by** (*auto simp: prog-def*)

lemma *is-Fence-pcOf*[simp]: $\text{pcOf } cfg < 6 \implies (\text{prog!}(\text{pcOf } cfg)) \neq \text{Fence}$
using *cases-5*[of *pcOf cfg*] **by** (*auto simp: prog-def*)

lemma *prog0*[simp]: $\text{prog } ! 0 = \text{Start}$
by (*auto simp: prog-def*)

lemma *prog1*[simp]: $\text{prog } ! (\text{Suc } 0) = \text{Input } U \text{ } xx$
by (*auto simp: prog-def*)

lemma *prog2*[simp]: $\text{prog } ! 2 = tt ::= (N 0)$
by (*auto simp: prog-def*)

lemma *prog3*[simp]: $\text{prog } ! 3 = \text{IfJump } (\text{Less } (V \text{ } xx) (N \text{ } NN)) 4 5$
by (*auto simp: prog-def*)

lemma *prog4*[simp]: $\text{prog } ! 4 = tt ::= (VA \text{ } aa2 (\text{Times } (VA \text{ } aa1 (V \text{ } xx)) (N \text{ } 512)))$
by (*auto simp: prog-def*)

lemma *prog5*[simp]: $\text{prog } ! 5 = \text{Output } U (V \text{ } tt)$
by (*auto simp: prog-def*)

lemma *isSecV-pcOf*[simp]:
 $\text{isSec } V (cfg, \text{ibT}, \text{ibUT}) \longleftrightarrow \text{pcOf } cfg = 0$
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf*[simp]:
 $\text{isSec } O (pstate, cfg, cfs, \text{ibT}, \text{ibUT}, ls) \longleftrightarrow (\text{pcOf } cfg = 0 \wedge cfs = [])$
using *isSecO-def* **by** *simp*

lemma *getInputT-not*[simp]: $\text{pcOf } cfg < 6 \implies$
 $(\text{prog } ! \text{pcOf } cfg) \neq \text{Input } T \text{ } x$
apply(*cases cfg*) **subgoal for** *pc s* **using** *cases-5*[of *pcOf cfg*]
by (*auto simp: prog-def*) .

lemma *getActV-pcOf*[simp]:
 $\text{pcOf } cfg < 6 \implies$
 $\text{getAct } V (cfg, \text{ibT}, \text{ibUT}, ls) =$

(if pcOf cfg = 1 then lhd ibUT else \perp)
apply(subst getActV-simps) **unfolding** prog-def
apply simp
using getActV-simps not-is-getTrustedInput-getActV prog-def **by** auto

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
)
apply(subst getObsV-simps)
unfolding prog-def **apply** simp
using getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def
by (metis less-irrefl-nat)

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)
apply(subst getActO-simps)
apply(cases cfgs, auto)
unfolding prog-def
using getActV-simps getActV-pcOf prog-def **by** presburger

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
)
apply(subst getObsO-simps)
apply(cases cfgs, auto)
unfolding prog-def
using getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def
by (metis getObsV-pcOf)

lemma nextB-pc0[simp]:
 nextB (Config 0 s, ibT, ibUT) =
 (Config 1 s, ibT, ibUT)
apply(subst nextB-Start-Skip-Fence)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma *readLocs-pc0*[simp]:
 $readLocs (Config\ 0\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1*[simp]:
 $ibUT \neq LNil \implies nextB (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p),\ ibT,\ ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1*[simp]:
 $readLocs (Config\ 1\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1'*[simp]:
 $ibUT \neq LNil \implies nextB (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p),\ ibT,\ ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1'*[simp]:
 $readLocs (Config\ (Suc\ 0)\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc2*[simp]:
 $nextB (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $((Config\ 3\ (State\ (Vstore\ (vs(tt := 0))))\ avst\ h\ p),\ ibT,\ ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc2*[simp]:
 $readLocs (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p)) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3-then*[simp]:
 $vs\ xx < NN \implies$
 $nextB (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
apply(*subst nextB-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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

(*Config 5 (State (Vstore vs) avst h p), ibT, ibUT*)
apply(subst nextB-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

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*)
by(cases vs xx < NN, auto)

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*)
apply(subst nextM-IfTrue)
unfolding endPC-def **unfolding** prog-def **by** auto

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*)
apply(subst nextM-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

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*)
by(cases vs xx < NN, auto)

lemma readLocs-pc3[simp]:
readLocs (*Config 3 s*) = {}
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

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)
apply(subst nextB-Assign)
unfolding endPC-def **unfolding** prog-def **by** auto

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})
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

lemma *nextB-pc5*[simp]:
nextB (*Config 5 s, ibT, ibUT*) = (*Config 6 s, ibT, ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc5*[simp]:
readLocs (*Config 5 (State (Vstore vs) avst (Heap h) p)*) =
 {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-stepB-pc*:
 $pc < 6 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_B nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-5*[*of pc*] **apply** *safe*
subgoal by *simp*
subgoal by *simp*

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def, metis llist.collapse*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal by(*cases vs xx < NN, simp-all*)

subgoal by(*cases vs xx < NN, simp-all*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

by *simp+ . .*

lemma *not-finalB*:
 $pc < 6 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$

$\neg \text{finalB } (\text{Config } pc \ s, \text{ibT}, \text{ibUT})$
using *nextB-stepB-pc* **by** (*simp add: stepB-iff-nextB*)

lemma *finalB-pc-iff'*:

$pc < 6 \implies$

$\text{finalB } (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \longleftrightarrow$
 $(pc = 1 \wedge \text{ibUT} = \text{LNil})$

subgoal apply *safe*

subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)

subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)

subgoal using *finalB-iff* **by** *auto . .*

lemma *finalB-pc-iff*:

$pc \leq 6 \implies$

$\text{finalB } (\text{Config } pc \ s, \text{ibT}, \text{ibUT}) \longleftrightarrow$
 $(pc = 1 \wedge \text{ibUT} = \text{LNil} \vee pc = 6)$

using *cases-5[of pc]* **apply** (*elim disjE, simp add: finalB-def*)

subgoal by (*meson final-def stebB-0*)

by (*simp add: finalB-pc-iff' finalB-endPC*)**+**

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

$pcOf \text{ cfg} \leq 6 \implies$

$\text{finalB } (\text{cfg}, \text{ibT}, \text{ibUT}) \longleftrightarrow (pcOf \text{ cfg} = 1 \wedge \text{ibUT} = \text{LNil} \vee pcOf \text{ cfg} = 6)$

by (*metis config.collapse finalB-pc-iff*)

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

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

lemma *vs-xx-cases:vs_i-t cfg \vee vs_i-f cfg* **unfolding** *vs_i-t-def vs_i-f-def* **by** *auto*

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

lemma *bool-invar[simp]: $\neg vs_i\text{-t } (\text{Config } 6 \ s) \implies vs_i\text{-t } (\text{Config } 6 \ s) \implies (\text{Config } 6 \ s, \text{ib1}) \rightarrow_B (\text{Config } 6 \ s, \text{ib1}) \implies \text{False}$*

unfolding *vs_i-defs*

by *simp*

lemma *nextB-vs-consistent-aux*:

$2 \leq pc \wedge pc < 6 \implies$

$(\text{nextB } (\text{Config } pc \ (\text{State } (\text{Vstore } vs) \text{ avst } (\text{Heap } h) \ p), \text{ibT}, \text{ibUT})) = (\text{Config } pc'$
 $(\text{State } (\text{Vstore } vs') \text{ avst}' (\text{Heap } h') \ p'), \text{ibT}', \text{ibUT}') \implies$

$\text{avst} = \text{avst}' \wedge$

$vs \text{ xx} = vs' \text{ xx} \wedge$

$h = h' \wedge$

$pc < pc'$

using *cases-5[of pc]* **apply**(*elim disjE*) **apply** *simp-all*

subgoal by *auto*

subgoal using *xx-NN-cases[of vs]* **by**(*elim disjE, simp-all*)

by *auto*

lemma *nextB-vs-consistent*:

$2 \leq pcOf\ cf\ g \wedge pcOf\ cf\ g < 6 \implies$
 $(nextB\ (cfg,\ ibT,\ ibUT)) = (cfg',\ ibT',\ ibUT') \implies$
 $(getAvstore\ (stateOf\ cf\ g)) = (getAvstore\ (stateOf\ cf\ g')) \wedge$
 $(getHheap\ (stateOf\ cf\ g)) = (getHheap\ (stateOf\ cf\ g')) \wedge$
 $vstore\ (getVstore\ (stateOf\ cf\ g))\ xx = vstore\ (getVstore\ (stateOf\ cf\ g'))\ xx$
apply(cases *cfg*) **subgoal for** *pc s*
apply(cases *s*) **subgoal for** *vstore avst heap-h p*
apply (cases *heap-h*, cases *vstore*, cases *avst*) **subgoal for** *h vs*
apply(cases *cfg'*) **subgoal for** *pc' s'*
apply(cases *s'*) **subgoal for** *vstore' avst'' heap-h' p'*
apply (cases *heap-h'*, cases *vstore'*, cases *avst''*) **subgoal for** *h vs*
using *nextB-vs-consistent-aux* **apply** *simp*
by *blast*

lemma *nextB-vs_i-t-consistent*:

$2 \leq pcOf\ cf\ g \wedge pcOf\ cf\ g < 6 \implies$
 $(nextB\ (cfg,\ ibT,\ ibUT)) = (cfg',\ ibT',\ ibUT') \implies$
 $vs_{i-t}\ cf\ g \longleftrightarrow vs_{i-t}\ cf\ g'$
unfolding *vs_i-defs* **using** *nextB-vs-consistent*
by *simp*

lemma *nextB-vs_i-f-consistent*:

$2 \leq pcOf\ cf\ g \wedge pcOf\ cf\ g < 6 \implies$
 $(nextB\ (cfg,\ ibT,\ ibUT)) = (cfg',\ ibT',\ ibUT') \implies$
 $vs_{i-f}\ cf\ g \longleftrightarrow vs_{i-f}\ cf\ g'$
unfolding *vs_i-defs* **using** *nextB-vs-consistent*
by *simp*

end

8.2 Proof

theory *Fun1-insecure*

imports *Fun1*

begin

8.2.1 Concrete leak

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

definition *same-xx* *cfg3* *cfgs3* *cfg4* *cfgs4* \equiv

$vstore\ (getVstore\ (stateOf\ cf\ g3))\ xx = vstore\ (getVstore\ (stateOf\ cf\ g4))\ xx \wedge$
 $(\forall\ cf\ g3' \in set\ cf\ gs3.\ vstore\ (getVstore\ (stateOf\ cf\ g3'))\ xx = vstore\ (getVstore\ (stateOf\ cf\ g3))\ xx) \wedge$
 $(\forall\ cf\ g4' \in set\ cf\ gs4.\ vstore\ (getVstore\ (stateOf\ cf\ g4'))\ xx = vstore\ (getVstore\ (stateOf\ cf\ g4))\ xx)$

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 $ss-neq-aux1 : nat(5 * 512) \neq nat(6 * 512)$ **by** *auto*

lemma $ss-neq-aux2 : nat(3 * 512) \neq nat(5 * 512)$ **by** *auto*

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 $aa2_{vs_3} \equiv array-loc\ aa2\ (nat\ (hh_3\ aa1_i * 512))\ avst'$

definition $vs_{33} = vs_2(tt := hh_3\ aa2_{vs_3})$

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 $(s\mathcal{3}_0:: \text{stateO}) \equiv (pstate_0, (\text{Config } 0 \text{ } s_{03}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s\mathcal{3}_1:: \text{stateO}) \equiv (pstate_0, (\text{Config } 1 \text{ } s_{03}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s\mathcal{3}_2:: \text{stateO}) \equiv (pstate_0, (\text{Config } 2 \text{ } s_{13}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s\mathcal{3}_3:: \text{stateO}) \equiv (pstate_0, (\text{Config } 3 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s\mathcal{3}_4:: \text{stateO}) \equiv (pstate_1, (\text{Config } 5 \text{ } s_{23}), [\text{Config } 4 \text{ } s_{23}], \text{repeat } (NN+1), \text{repeat } (NN+1), \{\})$

definition $(s\mathcal{3}_5:: \text{stateO}) \equiv (pstate_1, (\text{Config } 5 \text{ } s_{23}), [\text{Config } 5 \text{ } s_{33}], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

definition $(s\mathcal{3}_6:: \text{stateO}) \equiv (pstate_2, (\text{Config } 5 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

definition $(s\mathcal{3}_7:: \text{stateO}) \equiv (pstate_2, (\text{Config } 6 \text{ } s_{23}), [], \text{repeat } (NN+1), \text{repeat } (NN+1), \{aa2_{vs3}, aa1_i\})$

lemmas $s\mathcal{3}\text{-def} = s\mathcal{3}_0\text{-def } s\mathcal{3}_1\text{-def } s\mathcal{3}_2\text{-def } s\mathcal{3}_3\text{-def } s\mathcal{3}_4\text{-def } s\mathcal{3}_5\text{-def } s\mathcal{3}_6\text{-def } s\mathcal{3}_7\text{-def}$

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

definition $s\mathcal{3}\text{-trans} \equiv [s\mathcal{3}_0, s\mathcal{3}_1, s\mathcal{3}_2, s\mathcal{3}_3, s\mathcal{3}_4, s\mathcal{3}_5, s\mathcal{3}_6, s\mathcal{3}_7]$

lemmas $s\mathcal{3}\text{-trans}\text{-defs} = s\mathcal{3}\text{-trans}\text{-def } s\mathcal{3}\text{-def}$

lemma *hd-s3-trans[simp]*: *hd s3-trans = s3₀* **by** (*simp add: s3-trans-def*)
lemma *s3-trans-nemp[simp]*: *s3-trans ≠ []* **by** (*simp add: s3-trans-def*)

lemma *s3₀₁[simp]*: *s3₀ →S s3₁*
unfolding *s3-def*
using *nonspec-normal*
by *simp*

lemma *s3₁₂[simp]*: *s3₁ →S s3₂*
unfolding *s3-def state-def*
using *nonspec-normal*
by *simp*

lemma *s3₂₃[simp]*: *s3₂ →S s3₃*
unfolding *s3-def state-def*
by (*simp add: finalM-iff*)

lemma *s3₃₄[simp]*: *s3₃ →S s3₄*
unfolding *s3-def state-def*
using *nonspec-mispred*
by (*simp add: finalM-iff*)

lemma *s3₄₅[simp]*: *s3₄ →S s3₅*
unfolding *s3-def state-def*
using *spec-normal*
by (*simp-all add: finalM-iff, blast*)

lemma *s3₅₆[simp]*: *s3₅ →S s3₆*
unfolding *s3-def state-def*
using *spec-resolve*
by *simp*

lemma *s3₆₇[simp]*: *s3₆ →S s3₇*
unfolding *s3-def state-def*
using *nonspec-normal*
by *simp*

lemma *finalS-s3₇[simp]*: *finalS s3₇*
unfolding *finalS-def final-def s3-def*
by (*simp add: stepS-endPC*)

lemmas *s3-trans-simps = s3₀₁ s3₁₂ s3₂₃ s3₃₄ s3₄₅ s3₅₆ s3₆₇*

definition (*s4₀:: stateO*) ≡ (*pstate₀, (Config 0 s₀₄), [], repeat (NN+1), repeat (NN+1), {}*)

definition (*s4₁:: stateO*) ≡ (*pstate₀, (Config 1 s₀₄), [], 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 $hd-s4-trans[simp]: hd\ s4-trans = s4_0$ **by** $(simp\ add: s4-trans-def)$
lemma $s4-trans-nemp[simp]: s4-trans \neq []$ **by** $(simp\ add: s4-trans-def)$

lemma $s4_{01}[simp]: s4_0 \rightarrow_S s4_1$
unfolding $s4-def$
using $nonspec-normal$
by $simp$

lemma $s4_{12}[simp]: s4_1 \rightarrow_S s4_2$
unfolding $s4-def\ state-def$
using $nonspec-normal$
by $simp$

lemma $s4_{24}[simp]: s4_2 \rightarrow_S s4_3$
unfolding $s4-def\ state-def$
using $nonspec-normal$
by $(simp\ add: finalM-iff)$

lemma $s4_{34}[simp]: s4_3 \rightarrow_S s4_4$
unfolding $s4-def\ state-def$
using $nonspec-mispred$
by $(simp\ add: finalM-iff)$

lemma $s4_{45}[simp]: s4_4 \rightarrow_S s4_5$
unfolding $s4-def\ state-def$
using $spec-normal$

by (*simp add: finalM-iff, blast*)

lemma $s4_{56}[simp]:s4_5 \rightarrow S s4_6$
unfolding *s4-def state-def*
using *spec-resolve*
by *simp*

lemma $s4_{67}[simp]:s4_6 \rightarrow S s4_7$
unfolding *s4-def state-def*
using *nonspec-normal*
by *simp*

lemma $finalS-s4_7[simp]:finalS s4_7$
unfolding *finalS-def final-def s4-def*
by (*simp add: stepS-endPC*)

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 $validS-s3-trans[simp]:Opt.validS s3-trans$
unfolding *Opt.validS-def validTransO.simps s3-trans-def*
apply *safe*
subgoal for i **using** *cases-5[of i]*
by(*elim disjE, simp-all*) .

lemma $validS-s4-trans[simp]:Opt.validS s4-trans$
unfolding *Opt.validS-def validTransO.simps s4-trans-def*
apply *safe*
subgoal for i **using** *cases-5[of i]*
by(*elim disjE, simp-all*) .

lemma $finalS-s3[simp]:finalS (last s3-trans)$ **by** (*simp add: s3-trans-def*)

lemma $finalS-s4[simp]:finalS (last s4-trans)$ **by** (*simp add: s4-trans-def*)

lemma $filter-s3[simp]:(filter isIntO (butlast s3-trans)) = (butlast s3-trans)$
unfolding *s3-trans-def finalS-def final-def*
using *s3-trans-simps validTransO.simps validTransO-iff-nextS*
by (*smt (verit) butlast.simps(2) filter.simps(1,2) isIntO.elims(3)*)

lemma $filter-s4[simp]:(filter isIntO (butlast s4-trans)) = (butlast s4-trans)$
unfolding *s4-trans-def finalS-def final-def*
using *s4-trans-simps validTransO.simps validTransO-iff-nextS*
by (*smt (verit) butlast.simps(2) filter.simps(1,2) isIntO.elims(3)*)

lemma $S-s3-trans[simp]:Opt.S s3-trans = [s03]$

apply (*simp add: Opt.S-def filtermap-def*)
unfolding *s3-trans-defs* **by** *simp*

lemma *S-s4-trans[simp]: Opt.S s4-trans = [s04]*
apply (*simp add: Opt.S-def filtermap-def*)
unfolding *s4-trans-defs* **by** *simp*

lemma *finalB-noStep[simp]: $\wedge s1'. \text{finalB } (cfg1, ibT1, ibUT1) \implies (cfg1, ibT1, ibUT1, ls1) \rightarrow N s1' \implies \text{False}$*
unfolding *finalN-def final-def finalB-eq-finalN* **by** *auto*

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]: $\forall x. x \neq \text{Suc } NN \longrightarrow hh1'\ x = hh2'\ x \wedge hh1'\ x = 0 \implies hh2'\ NN = 0$*
by (*metis n-not-Suc-n*)

lemma *heap1-eq0[simp]: $\forall x. x \neq \text{Suc } NN \longrightarrow hh1'\ x = hh2'\ x \wedge hh1'\ x = 0 \implies vs2\ xx < NN \implies hh2'\ (\text{nat } (vs2\ xx)) = 0$*
using *le-less-Suc-eq nat-le-eq-zle nat-less-eq-zless*
by (*metis lessI nat-int order.asym*)

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-var-val* *tt* 0 *cfg1* *cfg2*) \wedge

 (*pcOf* *cfg1* > 1 \longrightarrow (*same-var* *xx* *cfg1* *cfg2*) \wedge
 (*vs_i-t* *cfg1* \longrightarrow *pcOf* *cfg1* \in *trueProg*) \wedge
 (*vs_i-f* *cfg1* \longrightarrow *pcOf* *cfg1* \in *falseProg*))
 \wedge
 ls1 = *ls2* \wedge

 pcOf *cfg1* \in *PC* \wedge

common-memory cfg1 cfg2
)

declare Γ -*inv.simps*[*simp del*]

lemmas Γ -*def* = Γ -*inv.simps*

lemmas Γ -*defs* = Γ -*def common-memory.simps PC-def aa1_i-def*
trueProg-def falseProg-def same-var-val-def same-var-def

lemma Γ -*implies*: Γ -*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-t cfg1*) \wedge

(*pcOf cfg2* = 4 \longrightarrow *vs_i-t cfg2*) \wedge

(*pcOf cfg1* > 1 \longrightarrow *vs_i-t cfg1* \longleftrightarrow *vs_i-t cfg2*) \wedge

(*finalB* (*cfg1,ibT1,ibUT1*) \longleftrightarrow *pcOf cfg1* = 6) \wedge

(*finalB* (*cfg2,ibT2,ibUT2*) \longleftrightarrow *pcOf cfg2* = 6)

unfolding Γ -*defs*

apply(*elim conjE, intro conjI*)

subgoal using *atLeastAtMost-iff* **by** *blast*

subgoal using *vs-xx-cases*[*of cfg2*] **by** (*elim disjE, simp-all*)

subgoal apply (*rule impI,simp*) **using** *vs-xx-cases*[*of cfg1*] **by** (*elim disjE, simp-all*)

subgoal apply (*rule impI,simp*) **using** *vs-xx-cases*[*of cfg2*] *vs_i-defs* **by** (*elim disjE, simp-all*)

subgoal by (*simp add: vs_i-defs*)

using *finalB-pcOf-iff*

apply (*metis atLeastAtMost-iff one-less-numeral-iff semiring-norm(76)*)

using *finalB-pcOf-iff*

by (*metis atLeastAtMost-iff numeral-One numeral-less-iff semiring-norm(76)*)

lemma *istateO-s3*[*simp*]:*istateO s3₀* **unfolding** *s3-def state-def* **by** *simp*

lemma *istateO-s4*[*simp*]:*istateO s4₀* **unfolding** *s4-def state-def* **by** *simp*

lemma *validFromS-s3*[*simp*]:*Opt.validFromS s3₀ s3-trans*

unfolding *Opt.validFromS-def* **by** *simp*

lemma *validFromS-s4*[*simp*]:*Opt.validFromS s4₀ s4-trans*

unfolding *Opt.validFromS-def* **by** *simp*

lemma *completedFromO-s3*[*simp*]:*completedFromO s3₀ s3-trans*

unfolding *Opt.completedFrom-def* **by** *simp*

lemma *completedFromO-s4*[simp]:*completedFromO s4₀ s4-trans*
unfolding *Opt.completedFrom-def* **by** *simp*

lemma *Act-eq*[simp]:*Opt.A s3-trans = Opt.A s4-trans*
apply (*simp add: Opt.A-def filtermap-def*)
unfolding *s3-trans-defs s4-trans-defs*
by *simp*

lemma *aa2-neq*:*aa2_{vs3} ≠ aa2_{vs4}*
unfolding *vs-def reads_m-def avst-defs h-def array-loc-def*
by (*simp add: avst-defs array-base-def split: if-splits*)

lemma *aa1-neq*:*aa2_{vs3} ≠ aa1_i*
apply(*rule notI*)
unfolding *vs-def reads_m-def avst-defs h-def array-loc-def*
by (*simp add: avst-defs array-base-def split: if-splits*)

lemma *aa1-neq2*:*aa2_{vs4} ≠ aa1_i*
apply(*rule notI*)
unfolding *vs-def reads_m-def avst-defs h-def array-loc-def*
by (*simp add: avst-defs array-base-def split: if-splits*)

lemma *Obs-neq*[simp]:*Opt.O s3-trans ≠ Opt.O s4-trans*
apply (*simp add: Opt.O-def filtermap-def*)
unfolding *s3-trans-def s4-trans-def* **apply** *clarsimp*
unfolding *s3-trans-defs s4-trans-defs* **apply** *simp*
using *aa2-neq aa1-neq aa1-neq2* **by** *blast*

lemma Γ -*init*[simp]: $\bigwedge s1 s2. \text{istateV } s1 \implies \text{corrState } s1 s3_0 \implies \text{istateV } s2 \implies$
 $\text{corrState } s2 s4_0 \implies \Gamma$ -*inv* *s1* [*s03*] *s2* [*s04*]
subgoal for *s1 s2* **apply**(*cases s1, cases s2, simp*)
unfolding *s3-def s4-def s-def h-def* **by** (*auto simp: Γ -defs*) .

lemma *val-neq-1*:*nat (hh2' (nat (vs2 xx)) * 512) ≠ 1*
by (*smt (z3) nat-less-eq-zless nat-one-as-int*)

lemma *unwindSD*[simp]:*Rel-Sec.unwindSDCond validTransV istateV isSecV get-*
SecV isIntV getIntV Γ -inv
unfolding *unwindSDCond-def*
proof(*intro allI, rule impI, elim conjE, intro conjI*)
fix *ss1 ss2 sl1 sl2*
assume *reachV ss1 reachV ss2*
and Γ : Γ -*inv* *ss1 sl1 ss2 sl2*

```

obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
note ss = ss1 ss2

obtain pc1 vs1 avst1 h1 p1 where
  cfg1: cfg1 = Config pc1 (State (Vstore vs1) avst1 h1 p1)
  by (cases cfg1) (metis state.collapse vstore.collapse)
obtain pc2 vs2 avst2 h2 p2 where
  cfg2: cfg2 = Config pc2 (State (Vstore vs2) avst2 h2 p2)
  by (cases cfg2) (metis state.collapse vstore.collapse)
note cfg = cfg1 cfg2

obtain hh1 where h1: h1 = Heap hh1 by(cases h1, auto)
obtain hh2 where h2: h2 = Heap hh2 by(cases h2, auto)
note hh = h1 h2

show isIntV ss1 = isIntV ss2
  using  $\Gamma$  unfolding isIntV.simps ss
  unfolding  $\Gamma$ -defs
  using vs-xx-cases[of cfg1]
  apply (elim disjE) by simp-all

then have finalB:finalB (cfg1, ibT1, ibUT1) = finalB (cfg2, ibT2, ibUT2)
  unfolding isIntV.simps finalN-iff-finalB ss by blast

show  $\neg$  isIntV ss1  $\longrightarrow$  move1  $\Gamma$ -inv ss1 sl1 ss2 sl2  $\wedge$  move2  $\Gamma$ -inv ss1 sl1
ss2 sl2
  apply(unfold ss, auto)
  subgoal unfolding move1-def finalB-defs by auto
  subgoal unfolding finalB
    unfolding move2-def finalB-defs by auto .

show isIntV ss1  $\longrightarrow$  getActV ss1 = getActV ss2  $\longrightarrow$  getObsV ss1 = getObsV
ss2  $\wedge$  move12  $\Gamma$ -inv ss1 sl1 ss2 sl2
  proof(unfold ss isIntV.simps finalN-iff-finalB, intro impI, rule conjI)
    assume final: $\neg$  finalB (cfg1, ibT1, ibUT1) and
      getAct:getActV (cfg1, ibT1, ibUT1, ls1) = getActV (cfg2, ibT2, ibUT2,
ls2)
    have not6:pc1 = 6  $\implies$  False
      using cfg final  $\Gamma$ 
      by simp

    show getObsV (cfg1, ibT1, ibUT1, ls1) = getObsV (cfg2, ibT2, ibUT2,
ls2)
      using  $\Gamma$  getAct unfolding ss
      apply–apply(frule  $\Gamma$ -implies, elim conjE)

```

```

using cases-5[of pcOf cfg1] cases-5[of pcOf cfg2]
by(elim disjE, simp-all add: Γ-defs final)

show move12 Γ-inv (cfg1, ibT1, ibUT1, ls1) sl1 (cfg2, ibT2, ibUT2, ls2)
sl2
unfolding move12-def validEtransO.simps
proof(intro allI, rule impI, elim conjE, unfold validTransV.simps isSecV-iff
getSecV.simps fst-conv)
fix ss1' ss2' sl1' sl2'

assume v: (cfg1, ibT1, ibUT1, ls1) →N ss1' (cfg2, ibT2, ibUT2, ls2)
→N ss2' and
sec: pcOf cfg1 ≠ 0 ∧ sl1 = sl1' ∨ pcOf cfg1 = 0 ∧ sl1 = stateOf cfg1
# sl1'
pcOf cfg2 ≠ 0 ∧ sl2 = sl2' ∨ pcOf cfg2 = 0 ∧ sl2 = stateOf cfg2
# sl2'

obtain cfg1' ibT1' ibUT1' ls1' where ss1': ss1' = (cfg1', ibT1', ibUT1',
ls1')
by (cases ss1', auto)
obtain cfg2' ibT2' ibUT2' ls2' where ss2': ss2' = (cfg2', ibT2', ibUT2',
ls2')
by (cases ss2', auto)

obtain pc1' vs1' avst1' h1' p1' where
cfg1': cfg1' = Config pc1' (State (Vstore vs1') avst1' h1' p1')
by (cases cfg1') (metis state.collapse vstore.collapse)
obtain pc2' vs2' avst2' h2' p2' where
cfg2': cfg2' = Config pc2' (State (Vstore vs2') avst2' h2' p2')
by (cases cfg2') (metis state.collapse vstore.collapse)
note cfg = cfg cfg1' cfg2'

obtain hh1' where h1': h1' = Heap hh1' by(cases h1', auto)
obtain hh2' where h2': h2' = Heap hh2' by(cases h2', auto)
note hh = hh h1' h2'

note ss = ss1 ss2 ss1' ss2'
have v':(cfg1, ibT1, ibUT1) →B (cfg1', ibT1', ibUT1') using v unfolding
ss by simp
then have v1:nextB (cfg1, ibT1, ibUT1) = (cfg1', ibT1', ibUT1') using
stepB-nextB by auto

have v'':(cfg2, ibT2, ibUT2) →B (cfg2', ibT2', ibUT2') using v unfolding
ss by simp
then have v2:nextB (cfg2, ibT2, ibUT2) = (cfg2', ibT2', ibUT2') using
stepB-nextB by auto
note valid = v' v1 v'' v2

have ls1':ls1' = ls1 ∪ readLocs cfg1 using v unfolding ss by simp

```

```

have  $ls2'.ls2' = ls2 \cup readLocs\ cfg2$  using  $v$  unfolding  $ss$  by  $simp$ 
note  $ls = ls1'\ ls2'$ 

```

```

note  $\Gamma\text{-simps} = cfg\ ls\ vs_i\text{-defs}\ hh\ array\text{-loc}\text{-def}$ 
       $array\text{-base}\text{-def}\ state\text{-def}\ PC\text{-def}$ 

```

```

show  $\Gamma\text{-inv}\ ss1'\ sl1'\ ss2'\ sl2'$ 
  using  $\Gamma\ valid\ getAct$ 
  unfolding  $ss$  apply–apply( $frule\ \Gamma\text{-implies}$ )
  using  $cases\text{-}5[of\ pc1]\ not6$  apply( $elim\ disjE,\ simp\text{-all}$ )
  unfolding  $\Gamma\text{-def}\ ss$ 
  prefer 4 subgoal using  $vs\text{-xx}\text{-cases}[of\ cfg1]$ 
  by ( $elim\ disjE,\ unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )
  subgoal by ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )
  subgoal by ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )
  subgoal by ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )
  subgoal using  $val\text{-neq}\text{-}1$  apply ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )

    using  $val\text{-neq}\text{-}1$  by ( $metis\ NN\text{-suc}\ add\text{-left}\text{-cancel}\ nat\text{-int}$ )
    subgoal by ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ )
    subgoal by ( $unfold\ \Gamma\text{-defs},\ auto\ simp\ add:\ \Gamma\text{-simps}$ ) .
qed
qed
qed

```

```

theorem  $\neg rsecure$ 
  apply( $rule\ unwindSD\text{-}rsecure[of\ s3_0\ s3\text{-trans}\ s4_0\ s4\text{-trans}\ \Gamma\text{-inv}]$ )
  by  $simp\text{-all}$ 

```

end

9 Proof of Relative Security for fun2

```

theory  $Fun2$ 
  imports
     $../Instance\text{-}IMP/Instance\text{-}Secret\text{-}IMem$ 
     $Relative\text{-}Security.Unwinding\text{-}fin$ 
begin

```

9.1 Function definition and Boilerplate

```

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

```

```

consts  $NN :: nat$ 
lemma  $NN: NN \geq 0$  by  $auto$ 

```

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 ≠ *aa2* *aa1* ≠ *xx* *aa1* ≠ *tt*

aa2 ≠ *aa1* *aa2* ≠ *xx* *aa2* ≠ *tt*

xx ≠ *aa1* *xx* ≠ *aa2* *xx* ≠ *tt*

tt ≠ *aa1* *tt* ≠ *aa2* *tt* ≠ *xx*

unfolding *vvars-defs* **by** *auto*

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

lemma *aa1*: *size-aa1* ≥ 0 **and** *aa2*:*size-aa2* ≥ 0 **by** *auto*

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

initAvstore (*Avstore as*) = (*as aa1* = (0, *nat size-aa1*) ∧ *as aa2* = (*nat size-aa1*,
nat 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 6 ,  
  / Fence ,  
  / tt ::= (VA aa2 (Times (VA aa1 (V xx)) (N 512))),  
  / Output U (V tt)  
]
```

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

apply(*cases i, simp-all*)

subgoal for *i* **apply**(*cases i, simp-all*)

subgoal for *i* **apply**(*cases i, simp-all*)

subgoal for *i* **apply**(*cases i, simp-all*)

subgoal for *i* **apply**(*cases i, simp-all*)

subgoal for *i* **apply**(*cases i, simp-all*)

subgoal for i apply(cases i , simp-all)

.....

lemma *xx-NN-cases*: $vs\ xx < int(NN) \vee vs\ xx \geq int(NN)$ **by** auto

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

$pcOf\ cfg < 6 \implies is-IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 3$

apply(cases cfg) **subgoal for** $pc\ s$ **using** cases-6[of $pcOf\ cfg$]

by (auto simp: prog-def) .

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

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

using cases-6[of pc]

by (auto simp: prog-def)

lemma *eq-Fence-pc*[simp]:

$pc < 6 \implies prog\ !\ pc = Fence \longleftrightarrow pc = 4$

using cases-6[of pc]

by (auto simp: prog-def)

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**

$istate = istate$

apply(unfold-locales)

unfolding prog-def **by** auto

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 $x \rightarrow B^* y == \text{star } \text{stepB } x y$

abbreviation

$\text{stepM-abbrev} :: \text{config} \times \text{val llist} \times \text{val llist} \Rightarrow \text{config} \times \text{val llist} \times \text{val llist} \Rightarrow$
 $\text{bool} (\text{infix } \langle \rightarrow M \rangle 55)$

where $x \rightarrow M y == \text{stepM } x y$

abbreviation

$\text{stepN-abbrev} :: \text{config} \times \text{val llist} \times \text{val llist} \times \text{loc set} \Rightarrow \text{config} \times \text{val llist} \times \text{val}$
 $\text{llist} \times \text{loc set} \Rightarrow \text{bool} (\text{infix } \langle \rightarrow N \rangle 55)$

where $x \rightarrow N y == \text{stepN } x y$

abbreviation

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

where $x \rightarrow N^* y == \text{star } \text{stepN } x y$

abbreviation

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

where $x \rightarrow S y == \text{stepS } x y$

abbreviation

$\text{stepsS-abbrev} :: \text{configS} \Rightarrow \text{configS} \Rightarrow \text{bool} (\text{infix } \langle \rightarrow S^* \rangle 55)$

where $x \rightarrow S^* y == \text{star } \text{stepS } x y$

lemma $\text{endPC}[simp]: \text{endPC} = 7$

unfolding endPC-def **unfolding** prog-def **by** auto

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

using $\text{cases-6}[of \text{pcOf } \text{cfg}]$ **by** $(\text{auto } simp: \text{prog-def})$

lemma $\text{start}[simp]: \text{prog} ! 0 = \text{Start}$

by $(\text{auto } simp: \text{prog-def})$

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

by $(\text{auto } simp: \text{prog-def})$

lemma $\text{if-stat}[simp]: \text{prog} ! 3 = (\text{IfJump } (\text{Less } (V \text{ } xx) (N \text{ } NN)) 4 6)$

by $(\text{auto } simp: \text{prog-def})$

lemma $\text{isOutput1}[simp]: \text{prog} ! 6 = \text{Output } U (V \text{ } tt)$

by $(\text{auto } simp: \text{prog-def})$

lemma *is-Output-pcOf[simp]*: $pcOf\ cfg < 7 \implies is-Output\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 6$
using *cases-6*[of *pcOf cfg*] **by** (*auto simp: prog-def*)

lemma *is-Fence-pcOf[simp]*: $pcOf\ cfg < 7 \implies (prog!(pcOf\ cfg)) = Fence \longleftrightarrow pcOf\ cfg = 4$
using *cases-6*[of *pcOf cfg*] **by** (*auto simp: prog-def*)

lemma *is-Output[simp]*: $is-Output\ (prog\ !\ 6)$
unfolding *is-Output-def prog-def* **by** *auto*

lemma *isSecV-pcOf[simp]*:
 $isSecV\ (cfg, ibT, ibUT) \longleftrightarrow pcOf\ cfg = 0$
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf[simp]*:
 $isSecO\ (pstate, cfg, cfs, ibT, ibUT, ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
using *isSecO-def* **by** *simp*

lemma *getInputT-not[simp]*: $pcOf\ cfg < 7 \implies (prog\ !\ pcOf\ cfg) \neq Input\ T\ inp$
apply(*cases cfg*) **subgoal for** *pc s* **using** *cases-6*[of *pcOf cfg*]
by (*auto simp: prog-def*) .

lemma *getActV-pcOf[simp]*:
 $pcOf\ cfg < 7 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1\ then\ lhd\ ibUT\ else\ \perp)$
apply(*subst getActV-simps*) **unfolding** *prog-def*
using *cases-6*[of *pcOf cfg*] **by** *auto*

lemma *getObsV-pcOf[simp]*:
 $pcOf\ cfg < 7 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 6\ then$
 $\ (outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $\ else\ \perp$
 $\)$
apply(*subst getObsV-simps*)
using *getObsV-simps not-is-Output-getObsV is-Output-pcOf*
unfolding *prog-def* **by** *simp*

lemma *getActO-pcOf[simp]*:
 $pcOf\ cfg < 7 \implies$
 $getActO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
(if $pcOf\ cfg = 1 \wedge cfgs = []$ *then* $lhd\ ibUT$ *else* \perp *)*
apply(*subst* *getActO-simps*)
apply(*cases* *cfgs*, *auto*)
unfolding *prog-def* **apply** *simp*
using *getActV-simps* *getActV-pcOf* *prog-def* **by** *presburger*

lemma *getObsO-pcOf[simp]*:
 $pcOf\ cfg < 7 \implies$
 $getObsO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
(if $pcOf\ cfg = 6 \wedge cfgs = []$ *then*
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg), ls)$
else \perp
)
apply(*subst* *getObsO-simps*)
apply(*cases* *cfgs*, *auto*)
using *getObsV-simps* *is-Output-pcOf* *not-is-Output-getObsV*
unfolding *prog-def* **by** *auto*

lemma *eqSec-pcOf[simp]*:
 $eqSec\ (cfg1, ibT, ibUT1, ls1)\ (pstate3, cfg3, cfgs3, ibT, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfgs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
unfolding *eqSec-def* **by** *simp*

lemma *nextB-pc0[simp]*:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
apply(*subst* *nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc0'[simp]*: $nextB\ (Config\ 0\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
 $=$
 $(Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT)$
apply(*subst* *nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc0[simp]*:

$readLocs (Config\ 0\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1[simp]*:
 $ibUT \neq LNil \implies nextB (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p),\ ibT,\ ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1[simp]*:
 $readLocs (Config\ 1\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1'[simp]*:
 $ibUT \neq LNil \implies nextB (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p),\ ibT,\ ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1'[simp]*:
 $readLocs (Config\ (Suc\ 0)\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc2[simp]*:
 $nextB (Config\ 2\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 3\ (State\ (Vstore\ (vs(tt := 0))))\ avst\ h\ p),\ ibT,\ ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc2[simp]*:
 $readLocs (Config\ 2\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3-then[simp]*:
 $vs\ xx < NN \implies$
 $nextB (Config\ 3\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 4\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
apply(*subst nextB-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3-else[simp]*:
 $vs\ xx \geq NN \implies$

$nextB (Config\ 3 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 6 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
apply(subst nextB-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma nextB-pc3:
 $nextB (Config\ 3 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ xx < NN\ then\ 4\ else\ 6) (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
by(cases vs xx < NN, auto)

lemma nextM-pc3-then[simp]:
 $vs\ xx \geq NN \implies$
 $nextM (Config\ 3 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 4 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
apply(subst nextM-IfTrue)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma nextM-pc3-else[simp]:
 $vs\ xx < NN \implies$
 $nextM (Config\ 3 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ 6 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
apply(subst nextM-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma nextM-pc3:
 $nextM (Config\ 3 (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ xx < NN\ then\ 6\ else\ 4) (State (Vstore\ vs)\ avst\ h\ p),\ ibT,\ ibUT)$
by(cases vs xx < NN, auto)

lemma readLocs-pc3[simp]:
 $readLocs (Config\ 3\ s) = \{\}$
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

lemma nextB-pc4[simp]:
 $nextB (Config\ 4\ s,\ ibT,\ ibUT) = (Config\ 5\ s,\ ibT,\ ibUT)$
apply(subst nextB-Start-Skip-Fence)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma readLocs-pc4[simp]:
 $readLocs (Config\ 4\ s) = \{\}$
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

lemma nextB-pc5[simp]:
 $nextB (Config\ 5 (State (Vstore\ vs)\ avst (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $(let\ l = (array-loc\ aa2 (nat (h (array-loc\ aa1 (nat (vs\ xx))\ avst) * 512))\ avst)$
 $in (Config\ 6 (State (Vstore (vs(tt := h\ l)))\ avst (Heap\ h)\ p)),\ ibT,\ ibUT)$

apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc6[simp]*:
nextB (Config 6 s, ibT, ibUT) = (Config 7 s, ibT, ibUT)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc6[simp]*:
readLocs (Config 6 (State (Vstore vs) avst (Heap h) p)) =
{}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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)
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-6[of pc]* **apply** *safe*
subgoal by *simp*
subgoal by *simp*

subgoal apply *simp* **apply**(*subst stepB.simps, unfold endPC-def*)
by (*simp add: prog-def, metis llist.exhaust-sel*)
subgoal apply *simp* **apply**(*subst stepB.simps, unfold endPC-def*)
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps, unfold endPC-def*)
by (*simp add: prog-def*)

subgoal by(*cases vs xx < NN, simp-all*)
subgoal by(*cases vs xx < NN, simp-all*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*

by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
by *simp+ . .*

lemma not-finalB:
 $pc < \gamma \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
using *nextB-stepB-pc* **by** (*simp add: stepB-iff-nextB*)

lemma finalB-pc-iff':
 $pc < \gamma \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil)$
subgoal apply safe
subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)
subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)
subgoal using *finalB-iff getUntrustedInput-pcOf* **by** *auto . .*

lemma finalB-pc-iff:
 $pc \leq \gamma \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil \vee pc = \gamma)$
using *cases-6[of pc]* **apply** (*elim disjE, simp add: finalB-def*)
subgoal by (*meson final-def stebB-0*)
by (*simp add: finalB-pc-iff' finalB-endPC*)+

lemma finalB-pcOf-iff[simp]:
 $pcOf\ cfg \leq \gamma \implies$
 $finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = \gamma)$
by (*metis config.collapse finalB-pc-iff*)

lemma finalS-cond:pcOf cfg < $\gamma \implies$ cfigs = [] \implies (pcOf cfg = 1 \longrightarrow ibUT \neq LNil) \implies $\neg finalS (pstate, cfg, cfigs, ibT, ibUT, ls)$
apply(*cases cfg*)
subgoal for *pc s* **apply**(*cases s*)
subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-6[of pc]* **apply**(*elim disjE*) **unfolding** *finalS-defs*
subgoal using *nonspec-normal[of [] Config pc (State (Vstore vs) avst hh p)*
pstate pstate ibT ibUT

Config 1 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]
using is-If-pc by force

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst*
hh p)

pstate pstate ibT ibUT
Config 2 (State (Vstore (vs(xx:= lhd ibUT))) avst hh

p)
ibT ltl ibUT [] ls ∪ readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst*
hh p)

pstate pstate ibT ibUT
Config 3 (State (Vstore (vs(tt:= 0))) avst hh p)
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]
prefer 7 subgoal by metis by simp-all

subgoal apply(*cases mispred pstate [3]*)
subgoal apply(*frule nonspec-mispred*[of cfigs *Config pc (State (Vstore vs) avst*
hh p)

pstate update pstate [pcOf (Config pc (State

(Vstore vs) avst hh p))]
ibT ibUT Config (if vs xx < NN then 4 else 6)

(State (Vstore vs) avst hh p)
ibT ibUT Config (if vs xx < NN then 6 else 4)

(State (Vstore vs) avst hh p)
ibT ibUT [Config (if vs xx < NN then 6 else

4) (State (Vstore vs) avst hh p)]
ls ∪ readLocs (Config pc (State (Vstore vs)

avst hh p)) ls]
prefer 9 subgoal by metis by (simp add: finalM-iff)+

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst*
hh p)

pstate pstate ibT ibUT
Config (if vs xx < NN then 4 else 6) (State (Vstore

vs) avst hh p)
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]
prefer 7 subgoal by metis by simp-all .

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst*
hh p)

pstate pstate ibT ibUT

Config 5 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls]

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal[*of *cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate 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 hh p))
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore vs) avst

hh p)) ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal[*of *cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT
Config 7 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls]

prefer 7 subgoal by metis by simp-all

by simp-all . . .

lemma *finalS-cond-spec:*

pcOf cfg < 7 ⇒
(pcOf (last cfgs) = 4 ∧ pcOf cfg = 6) ∨ (pcOf (last cfgs) = 6 ∧ pcOf cfg =

4) ⇒

length cfgs = Suc 0 ⇒
¬ finalS (pstate, cfg, cfgs, ibT, ibUT, ls)

apply(*cases cfg*)

subgoal for *pc s apply*(*cases s*)

subgoal for *vst avst hh p apply*(*cases vst, cases avst, cases hh*)

subgoal for *vs as h*

apply(*elim disjE, elim conjE*) **unfolding** *finalS-defs*

subgoal apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)

subgoal using *spec-resolve[*of *cfgs pstate cfg update pstate (pcOf cfg # map*

pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls] by fastforce

subgoal using *spec-Fence[*of *cfgs pstate cfg pstate cfg [] ibT ibT ibUT ibUT*

ls ls] by fastforce .

subgoal apply(*elim conjE*)

apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)

subgoal using *spec-resolve[*of *cfgs pstate cfg update pstate (pcOf cfg # map*

pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls] by fastforce

subgoal using *spec-resolve[*of *cfgs pstate cfg update pstate (pcOf cfg # map*

pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls] by force

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

$(\text{getAvstore } (\text{stateOf } \text{cfg4})) \wedge$
 $\text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg3})) = \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg4})) \wedge$
 $(\forall \text{cfg3}' \in \text{set } \text{cfgs3}. \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg3}')) = \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg3}))) \wedge$
 $(\forall \text{cfg4}' \in \text{set } \text{cfgs4}. \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg4}')) = \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg4}))) \wedge$
 $///$
 $(\text{statA} = \text{Diff} \longrightarrow \text{statO} = \text{Diff}))$

lemma *common-implies: common* $(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT}, \text{ibUT2}, \text{ls2})$
 $\text{statO} \implies$
 $\text{pcOf } \text{cfg1} < 8 \wedge \text{pcOf } \text{cfg2} = \text{pcOf } \text{cfg1}$
unfolding *common-def PC-def*
by *(auto simp: image-def subset-eq)*

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**
 $\Delta 0 = (\lambda \text{num}$
 $(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO}.$
 $(\text{common } (\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO} \wedge$
 $\text{ibUT1} = \text{ibUT3} \wedge \text{ibUT2} = \text{ibUT4} \wedge$
 $(\text{pcOf } \text{cfg3} > 1 \longrightarrow \text{same-xx } \text{cfg3 } \text{cfgs3 } \text{cfg4 } \text{cfgs4}) \wedge$
 $(\text{pcOf } \text{cfg3} < 2 \longrightarrow \text{ibUT1} \neq \text{LNil} \wedge \text{ibUT2} \neq \text{LNil} \wedge \text{ibUT3} \neq \text{LNil} \wedge \text{ibUT4} \neq \text{LNil})$
 \wedge
 $\text{ls1} = \text{ls3} \wedge \text{ls2} = \text{ls4} \wedge$
 $\text{pcOf } \text{cfg3} \in \text{beforeInput} \wedge$
 $\text{noMisSpec } \text{cfgs3}$
 $))$

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

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 \longrightarrow ibUT3 \neq LNil) \wedge
 (pcOf cfg4 = 1 \longrightarrow ibUT4 \neq LNil) \wedge
 pcOf cfg1 < 7 \wedge pcOf cfg2 = pcOf cfg1 \wedge
 cfs3 = [] \wedge pcOf cfg3 < 7 \wedge
 cfs4 = [] \wedge pcOf cfg4 < 7
unfolding $\Delta 0$ -defs
apply (intro conjI)
apply simp-all
by (metis map-is-Nil-conv)

definition $\Delta 1 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**
 $\Delta 1 = (\lambda \text{num}$
 (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
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 same-xx cfg3 cfs3 cfg4 cfs4 \wedge
 pcOf cfg3 \in afterInput \wedge
 noMisSpec cfs3
))

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, 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

```

cfigs3 = [] ∧ pcOf cfig3 ≠ 1 ∧ pcOf cfig3 < 7 ∧
cfigs4 = [] ∧ pcOf cfig4 ≠ 1 ∧ pcOf cfig4 < 7
unfolding Δ1-defs
apply(intro conjI) apply simp-all
apply linarith
apply (metis list.map-disc-iff)
using semiring-norm(83,84)
by linarith

```

definition Δ2 :: enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status
⇒ bool **where**

```

Δ2 = (λnum
  (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 ∧
    ls1 = ls3 ∧ ls2 = ls4 ∧
    same-xx cfg3 cfgs3 cfg4 cfgs4 ∧
    pcOf cfig3 = startOfThenBranch ∧
    pcOf (last cfigs3) = elseBranch ∧
    misSpecL1 cfigs3
  ))

```

lemmas Δ2-defs = Δ2-def common-def PC-def same-xx-def inThenBranch-def
elseBranch-def startOfThenBranch-def misSpecL1-def same-xx-def

lemma Δ2-implies: Δ2 num (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
(pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
statA
(cfg1,ibT1,ibUT1,ls1)
(cfg2,ibT2,ibUT2,ls2)
statO ⇒
pcOf (last cfigs3) = 6 ∧ pcOf cfig3 = 4 ∧
pcOf (last cfigs4) = pcOf (last cfigs3) ∧
pcOf cfig3 = pcOf cfig4 ∧
length cfigs3 = Suc 0 ∧
length cfigs3 = length cfigs4
apply(intro conjI)
unfolding Δ2-defs **apply** simp-all
apply (simp add: image-subset-iff)

apply (*metis last-map list.map-disc-iff*)
by (*metis length-map*)

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*, *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* *cfgs3*) = *startOfThenBranch* \wedge
same-xx *cfg3* *cfgs3* *cfg4* *cfgs4* \wedge
misSpecL1 *cfgs3*
))

lemmas $\Delta 3\text{-defs} = \Delta 3\text{-def}$ *common-def* *PC-def* *same-xx-def* *elseBranch-def* *startOfThen-Branch-def*
misSpecL1-def *same-xx-def*

lemma $\Delta 3\text{-implies}$: $\Delta 3$ *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 \wedge *pcOf* *cfg3* = 6 \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*

apply(*intro conjI*)

unfolding $\Delta 3\text{-defs}$ **apply** *simp-all*

```

apply (simp add: image-subset-iff)
apply (metis last-map map-is-Nil-conv)
apply (metis last-in-set list.size(3) n-not-Suc-n)
apply (metis One-nat-def last-in-set length-0-conv length-map zero-neq-one)
by (metis length-map)

```

definition $\Delta_4 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

```

 $\Delta_4 = (\lambda \text{num}$ 
  (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
  (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
  statA
  (cfg1, ibT1, ibUT1, ls1)
  (cfg2, ibT2, ibUT2, ls2)
  statO.
  (pcOf cfg3 = endPC  $\wedge$  pcOf cfg4 = endPC  $\wedge$  cfs3 = []  $\wedge$  cfs4 = []  $\wedge$ 
  pcOf cfg1 = endPC  $\wedge$  pcOf cfg2 = endPC))

```

lemmas $\Delta_4\text{-defs} = \Delta_4\text{-def common-def endPC-def}$

lemma *init: initCond* Δ_0

```

unfolding initCond-def
unfolding initCond-def apply(intro allI)
subgoal for  $s_3 s_4$  apply(cases  $s_3$ , cases  $s_4$ )
subgoal for pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 pstate4 cfg4 cfs4 ibT4 ibUT4 ls4
apply clarsimp
apply(cases getAvstore (stateOf cfg3), cases getAvstore (stateOf cfg4))
unfolding  $\Delta_0\text{-defs}$ 
unfolding array-base-def by auto . .

```

lemma *step0: unwindIntoCond* Δ_0 (oor Δ_0 Δ_1)

```

proof(rule unwindIntoCond-simpleI)
fix  $n ss_3 ss_4 \text{statA} ss_1 ss_2 \text{statO}$ 
assume  $r: \text{reachO } ss_3 \text{ reachO } ss_4 \text{ reachV } ss_1 \text{ reachV } ss_2$ 
and  $\Delta_0: \Delta_0 n ss_3 ss_4 \text{statA} ss_1 ss_2 \text{statO}$ 

```

```

obtain pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 where  $ss_3: ss_3 = (\text{pstate3}, \text{cfg3}, \text{cfs3},$ 
  ibT3, ibUT3, ls3)
by (cases  $ss_3$ , auto)
obtain pstate4 cfg4 cfs4 ibT4 ibUT4 ls4 where  $ss_4: ss_4 = (\text{pstate4}, \text{cfg4}, \text{cfs4},$ 
  ibT4, ibUT4, ls4)
by (cases  $ss_4$ , auto)
obtain cfg1 ibT1 ibUT1 ls1 where  $ss_1: ss_1 = (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$ 
by (cases  $ss_1$ , auto)

```

obtain $cfg2\ ibT2\ ibUT2\ ls2$ **where** $ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)$
by $(cases\ ss2, auto)$
note $ss = ss3\ ss4\ ss1\ ss2$

obtain $pc3\ vs3\ avst3\ h3\ p3$ **where**
 $cfg3: cfg3 = Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)$
by $(cases\ cfg3)\ (metis\ state.collapse\ vstore.collapse)$
obtain $pc4\ vs4\ avst4\ h4\ p4$ **where**
 $cfg4: cfg4 = Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)$
by $(cases\ cfg4)\ (metis\ state.collapse\ vstore.collapse)$
note $cfg = cfg3\ cfg4$

obtain $hh3$ **where** $h3: h3 = Heap\ hh3$ **by** $(cases\ h3, auto)$
obtain $hh4$ **where** $h4: h4 = Heap\ hh4$ **by** $(cases\ h4, auto)$
note $hh = h3\ h4$

have $f1: \neg finalN\ ss1$
using $\Delta 0\ finalB-pc-iff'$ **unfolding** $ss\ finalN-iff-finalB\ \Delta 0-defs$
by $simp$

have $f2: \neg finalN\ ss2$
using $\Delta 0\ finalB-pc-iff'$ **unfolding** $ss\ finalN-iff-finalB\ \Delta 0-defs$
by $simp$

have $f3: \neg finalS\ ss3$
using $\Delta 0$ **unfolding** ss **apply-apply** $(frule\ \Delta 0-implies)$
using $finalS-cond$ **by** $simp$

have $f4: \neg finalS\ ss4$
using $\Delta 0$ **unfolding** ss **apply-apply** $(frule\ \Delta 0-implies)$
using $finalS-cond$ **by** $simp$

note $finals = f1\ f2\ f3\ f4$
show $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$
using $finals$ **by** $auto$

then show $isIntO\ ss3 = isIntO\ ss4$ **by** $simp$

show $react\ (oor\ \Delta 0\ \Delta 1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$
unfolding $react-def$ **proof** $(intro\ conjI)$

show $match1\ (oor\ \Delta 0\ \Delta 1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$
unfolding $match1-def$ **by** $(simp\ add: finalS-defs)$
show $match2\ (oor\ \Delta 0\ \Delta 1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$
unfolding $match2-def$ **by** $(simp\ add: finalS-defs)$
show $match12\ (oor\ \Delta 0\ \Delta 1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$

```

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain pc3 vs3 avst3 h3 p3 where
  cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
  cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  show eqSec ss1 ss3
  using v sa  $\Delta 0$  unfolding ss
  by (simp add:  $\Delta 0$ -defs eqSec-def)

  show eqSec ss2 ss4
  using v sa  $\Delta 0$  unfolding ss
  apply (simp add:  $\Delta 0$ -defs eqSec-def)
  by (metis length-0-conv length-map)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 0$  unfolding ss
  unfolding Opt.eqAct-def Van.eqAct-def
  apply(simp-all add:  $\Delta 0$ -defs)
  by (metis f3 map-is-Nil-conv ss3)

  show match12-12 (oor  $\Delta 0$   $\Delta 1$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
    show validTransV (ss1, nextN ss1)
      by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
      by (simp add: f2 nextN-stepN)

```

```

{assume sstat: statA' = Diff
 show sstatO' statO ss1 ss2 = Diff
 using v sa Δ0 sstat unfolding ss cfg statA' apply simp
 apply(simp add: Δ0-defs sstatO'-def sstatA'-def finalS-def final-def)
 using cases-6[of pc3] apply(elim disjE)
 apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
 apply(cases statO, simp-all) apply (cases statA, simp-all)
 apply (fastforce)
 using newStat.simps status.exhaust status.distinct by (smt(z3))
} note stat = this

have cfgs3-e:cfgs3 = [] using Δ0[unfolded Δ0-defs ss cfg] by fast

  show oor Δ0 Δ1 ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO' statO
ss1 ss2)

  using v3[unfolded ss, simplified] cfgs3-e proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa Δ0 stat unfolding ss apply (simp add:
Δ0-defs)
  by (metis is-If-pc less-Suc-eq nat-less-le numeral-1-eq-Suc-0 numeral-3-eq-3
one-eq-numeral-iff semiring-norm(83) zero-less-numeral zero-neq-numeral)

  next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis
  using v3[unfolded ss, simplified] cfgs3-e proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa Δ0 stat nn3 unfolding ss by (simp add:
Δ0-defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  show ?thesis using sa stat Δ0 v3 v4 nn3 nn4 f4 unfolding ss cfg hh
Opt.eqAct-def
  apply clarsimp
  using cases-6[of pc3] apply(elim disjE)
  subgoal apply(rule oorI1) by (simp add: Δ0-defs)
  subgoal apply(rule oorI2) apply (simp add: Δ0-defs,auto)
  unfolding Δ1-defs
  subgoal by (simp add: Δ0-defs)
  subgoal by (simp add: Δ0-defs) .
  by (simp add: Δ0-defs)+
  qed simp+
  qed simp+
  qed
  qed
  qed
  qed

```

```

lemma step1: unwindIntoCond  $\Delta 1$  (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ )
proof(rule unwindIntoCond-simpleI)
  fix  $n$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 
  assume  $r$ : reachO  $ss3$  reachO  $ss4$  reachV  $ss1$  reachV  $ss2$ 
  and  $\Delta 1$ :  $\Delta 1$   $n$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 

  obtain  $pstate3$   $cfg3$   $cfgs3$   $ibT3$   $ibUT3$   $ls3$  where  $ss3$ :  $ss3 = (pstate3, cfg3, cfgs3,$ 
 $ibT3, ibUT3, ls3)$ 
  by (cases  $ss3$ , auto)
  obtain  $pstate4$   $cfg4$   $cfgs4$   $ibT4$   $ibUT4$   $ls4$  where  $ss4$ :  $ss4 = (pstate4, cfg4, cfgs4,$ 
 $ibT4, ibUT4, ls4)$ 
  by (cases  $ss4$ , auto)
  obtain  $cfg1$   $ibT1$   $ibUT1$   $ls1$  where  $ss1$ :  $ss1 = (cfg1, ibT1, ibUT1, ls1)$ 
  by (cases  $ss1$ , auto)
  obtain  $cfg2$   $ibT2$   $ibUT2$   $ls2$  where  $ss2$ :  $ss2 = (cfg2, ibT2, ibUT2, ls2)$ 
  by (cases  $ss2$ , auto)
  note  $ss = ss3$   $ss4$   $ss1$   $ss2$ 

  obtain  $pc1$   $vs1$   $avst1$   $h1$   $p1$  where
 $cfg1$ :  $cfg1 = Config$   $pc1$  (State (Vstore  $vs1$ )  $avst1$   $h1$   $p1$ )
  by (cases  $cfg1$ ) (metis state.collapse vstore.collapse)
  obtain  $pc2$   $vs2$   $avst2$   $h2$   $p2$  where
 $cfg2$ :  $cfg2 = Config$   $pc2$  (State (Vstore  $vs2$ )  $avst2$   $h2$   $p2$ )
  by (cases  $cfg2$ ) (metis state.collapse vstore.collapse)
  obtain  $pc3$   $vs3$   $avst3$   $h3$   $p3$  where
 $cfg3$ :  $cfg3 = Config$   $pc3$  (State (Vstore  $vs3$ )  $avst3$   $h3$   $p3$ )
  by (cases  $cfg3$ ) (metis state.collapse vstore.collapse)
  obtain  $pc4$   $vs4$   $avst4$   $h4$   $p4$  where
 $cfg4$ :  $cfg4 = Config$   $pc4$  (State (Vstore  $vs4$ )  $avst4$   $h4$   $p4$ )
  by (cases  $cfg4$ ) (metis state.collapse vstore.collapse)
  note  $cfg = cfg1$   $cfg2$   $cfg3$   $cfg4$ 

  obtain  $hh3$  where  $h3$ :  $h3 = Heap$   $hh3$  by(cases  $h3$ , auto)
  obtain  $hh4$  where  $h4$ :  $h4 = Heap$   $hh4$  by(cases  $h4$ , auto)
  note  $hh = h3$   $h4$ 

  have  $f1$ :  $\neg finalN$   $ss1$ 
    using  $\Delta 1$  finalB-pc-iff' unfolding  $ss$   $cfg$  finalN-iff-finalB  $\Delta 1$ -defs
    by simp linarith

  have  $f2$ :  $\neg finalN$   $ss2$ 
    using  $\Delta 1$  finalB-pc-iff' unfolding  $ss$   $cfg$  finalN-iff-finalB  $\Delta 1$ -defs
    by simp linarith

  have  $f3$ :  $\neg finalS$   $ss3$ 
    using  $\Delta 1$  unfolding  $ss$  apply-apply(frule  $\Delta 1$ -implies)

```

```

using finalS-cond by simp

have f4:¬finalS ss4
  using Δ1 unfolding ss apply-apply(frule Δ1-implies)
  using finalS-cond by simp

note finals = f1 f2 f3 f4

show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react (oor4 Δ1 Δ2 Δ3 Δ4) ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 (oor4 Δ1 Δ2 Δ3 Δ4) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor4 Δ1 Δ2 Δ3 Δ4) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 (oor4 Δ1 Δ2 Δ3 Δ4) ss3 ss4 statA ss1 ss2 statO

  proof(rule match12-simpleI, rule disjI2, intro conjI)
    fix ss3' ss4' statA'
    assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

    show eqSec ss1 ss3
    using v sa Δ1 unfolding ss
    by (simp add: Δ1-defs eqSec-def)

    show eqSec ss2 ss4
    using v sa Δ1 unfolding ss
    apply (simp add: Δ1-defs eqSec-def)
    by (metis length-0-conv length-map)

    show Van.eqAct ss1 ss2
    using v sa Δ1 unfolding ss Van.eqAct-def
    apply (simp-all add: Δ1-defs)

```

```

by linarith

show match12-12 (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ ) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

{assume sstat: statA' = Diff
show sstatO' statO ss1 ss2 = Diff
using v sa  $\Delta 1$  sstat unfolding ss cfg statA'
apply(simp add:  $\Delta 1$ -defs sstatO'-def sstatA'-def)
using cases-6[of pc3] apply(elim disjE)
defer 1 defer 1
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss status.distinct(1) newStat.simps by auto
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss status.distinct(1) newStat.simps by auto
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss status.distinct(1) newStat.simps by auto
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss status.distinct(1) newStat.simps by auto
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss status.distinct(1) newStat.simps by auto
  by simp+
} note stat = this

have cfigs-e:cfigs3 = [] cfigs4 = [] using  $\Delta 1$ -implies[OF  $\Delta 1$ [unfolded ss]] by
auto

show (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ )  $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2)
(ssstatO' statO ss1 ss2)

using v3[unfolded ss, simplified] cfigs-e proof(cases rule: stepS-cases)
case nonspec-mispred note nm3 = nonspec-mispred
  show ?thesis using v4[unfolded ss, simplified] cfigs-e proof(cases rule:
stepS-cases)

    case nonspec-normal
    then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
  case nonspec-mispred note nm4 = nonspec-mispred
  then show ?thesis

```

```

using sa  $\Delta 1$  stat v3 v4 nm3 nm4 unfolding ss cfg hh apply clarsimp
using cases-6[of pc3] apply(elim disjE)
  subgoal by simp
  subgoal by simp
  subgoal by simp
  subgoal using xx-NN-cases[of vs3] apply(elim disjE)
    subgoal apply(rule oor4I2) by (simp add:  $\Delta 1$ -defs  $\Delta 2$ -defs)
    subgoal apply(rule oor4I3) by (simp add:  $\Delta 1$ -defs  $\Delta 3$ -defs) .
  by (simp add:  $\Delta 1$ -defs)+
qed simp+
next
case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4[unfolded ss, simplified] cfgs-e proof(cases rule:
stepS-cases)

    case nonspec-mispred
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
  case nonspec-normal
  then show ?thesis using sa  $\Delta 1$  stat v3 v4 nn3 unfolding ss cfg hh
apply clarsimp
  using cases-6[of pc3] apply(elim disjE)
  subgoal by (simp add:  $\Delta 1$ -defs)
  subgoal by (simp add:  $\Delta 1$ -defs)
  subgoal apply(rule oor4I1) by(simp add: $\Delta 1$ -defs)
  subgoal using xx-NN-cases[of vs3] apply(elim disjE)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs) .
  subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
  subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
  subgoal apply(rule oor4I4) by (simp add:  $\Delta 1$ -defs  $\Delta 4$ -defs)
  subgoal apply(rule oor4I4) by (simp add:  $\Delta 1$ -defs  $\Delta 4$ -defs) .
  qed simp+
  qed simp+
  qed
  qed
  qed
  qed

```

lemma step2: unwindIntoCond $\Delta 2$ $\Delta 1$

proof(rule unwindIntoCond-simpleI)

fix n ss3 ss4 statA ss1 ss2 statO

assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2

and $\Delta 2$: $\Delta 2$ n ss3 ss4 statA ss1 ss2 statO

obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 **where** ss3: ss3 = (pstate3, cfg3, cfgs3,

```

ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where
lcfgs3: last cfs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases last cfs3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
lcfgs4: last cfs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases last cfs4) (metis state.collapse vstore.collapse)
  note lcfgs = lcfgs3 lcfgs4

  have f1:¬finalN ss1
    using Δ2 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ2-defs
    by auto

  have f2:¬finalN ss2
    using Δ2 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ2-defs
    by auto

  have f3:¬finalS ss3
    using Δ2 unfolding ss apply-apply(frule Δ2-implies)
    using finalS-cond-spec by simp

  have f4:¬finalS ss4
    using Δ2 unfolding ss apply-apply(frule Δ2-implies)
    using finalS-cond-spec by simp

  note finals = f1 f2 f3 f4
  show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
    using finals by auto

  then show isIntO ss3 = isIntO ss4 by simp

  show react Δ1 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

    show match1 Δ1 ss3 ss4 statA ss1 ss2 statO
    unfolding match1-def by (simp add: finalS-def final-def)
    show match2 Δ1 ss3 ss4 statA ss1 ss2 statO
    unfolding match2-def by (simp add: finalS-def final-def)

```

```

show match12  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI1, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
  note hh = h3 h4

  show  $\neg$  isSecO ss3
  using v sa  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

  show  $\neg$  isSecO ss4
  using v sa  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

  show stat: statA = statA'  $\vee$  statO = Diff
  using v sa  $\Delta 2$ 
  apply (cases ss3, cases ss4, cases ss1, cases ss2)
  apply (cases ss3', cases ss4', clarsimp)
  using v sa  $\Delta 2$  unfolding ss statA' apply clarsimp
  apply(simp-all add:  $\Delta 2$ -defs sstatA'-def)
  apply(cases statO, simp-all)
  apply(cases statA, simp-all)
  unfolding finalS-def final-def
  by (smt (verit, ccfv-SIG) newStat.simps(1))

  have isO:is-Output (prog ! pcOf (last cfigs3)) is-Output (prog ! pcOf (last
  cfigs4)) using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by auto
  have pstate3:pstate3 = pstate4 using  $\Delta 2$ [unfolded ss  $\Delta 2$ -defs] by fast
  show  $\Delta 1 \infty$  ss3' ss4' statA' ss1 ss2 statO

  using v3[unfolded ss, simplified] isO proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

```

```

next
  case spec-normal
  then show ?thesis using sa stat  $\Delta 2$  v3 unfolding ss apply-
  apply(frule  $\Delta 2$ -implies) by(simp add:  $\Delta 2$ -defs)
next
  case spec-mispred
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply-
  apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
next
  case spec-Fence
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply-
  apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolveI
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply-
  apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using
 $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case spec-normal
  then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case spec-mispred
  then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case spec-Fence
  then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case spec-resolveI
  then show ?thesis using sa stat  $\Delta 2$  sr3 isO by blast
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs, metis)
next

```

```

      case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
      show ?thesis using sa stat Δ2 v3 v4 sr3 sr4
      unfolding ss lcfgs hh apply-
      apply(frule Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
    qed
  next
  case spec-resolveO note srO3 = spec-resolveO
    have cfgs4:cfgs4 ≠ [] using sa stat Δ2 unfolding ss apply-by(frule
Δ2-implies, auto)
    show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis by (simp add: cfgs4)
    next
      case nonspec-mispred
      then show ?thesis by (simp add: cfgs4)
    next
      case spec-normal
      then show ?thesis by (simp add: isO)
    next
      case spec-mispred
      then show ?thesis using isO by blast
    next
      case spec-Fence
      then show ?thesis using isO by blast
    next
      case spec-resolveI
      then show ?thesis using isO by blast
    next
      case spec-resolveO note srO4 = spec-resolveO
      show ?thesis using sa stat Δ2 v3 v4 srO3 srO4
      unfolding ss lcfgs hh apply-
      apply(frule Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
    next
      case spec-resolve note sr4 = spec-resolve(1,3-) isO(2)
      show ?thesis using sa stat Δ2 v3 v4 srO3 sr4
      unfolding ss lcfgs hh apply-
      apply(frule Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
    qed
  qed
qed
qed
qed

```

```

lemma step3: unwindIntoCond Δ3 (oor Δ3 Δ1)
proof(rule unwindIntoCond-simpleI)
  fix n ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2

```

and $\Delta 3$: $\Delta 3$ n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by $(cases\ ss3, auto)$

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by $(cases\ ss4, auto)$

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by $(cases\ ss1, auto)$

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by $(cases\ ss2, auto)$

note $ss = ss3\ ss4\ ss1\ ss2$

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$lcfgs3$: $last\ cfgs3 = Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)$

by $(cases\ last\ cfgs3)\ (metis\ state.collapse\ vstore.collapse)$

obtain $pc4$ $vs4$ $avst4$ $h4$ $p4$ **where**

$lcfgs4$: $last\ cfgs4 = Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)$

by $(cases\ last\ cfgs4)\ (metis\ state.collapse\ vstore.collapse)$

note $lcfgs = lcfgs3\ lcfgs4$

obtain $hh3$ **where** $h3$: $h3 = Heap\ hh3$ **by** $(cases\ h3, auto)$

obtain $hh4$ **where** $h4$: $h4 = Heap\ hh4$ **by** $(cases\ h4, auto)$

note $hh = h3\ h4$

have $f1$: $\neg finalN\ ss1$

using $\Delta 3\ finalB-pc-iff'$ **unfolding** $ss\ finalN-iff-finalB\ \Delta 3-defs$

by $auto$

have $f2$: $\neg finalN\ ss2$

using $\Delta 3\ finalB-pc-iff'$ **unfolding** $ss\ finalN-iff-finalB\ \Delta 3-defs$

by $auto$

have $f3$: $\neg finalS\ ss3$

using $\Delta 3\ unfolding\ ss\ apply-apply(frul\ \Delta 3-implies)$

using $finalS-cond-spec$ **by** $simp$

have $f4$: $\neg finalS\ ss4$

using $\Delta 3\ unfolding\ ss\ apply-apply(frul\ \Delta 3-implies)$

using $finalS-cond-spec$ **by** $simp$

note $finals = f1\ f2\ f3\ f4$

show $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$

using $finals$ **by** $auto$

```

then show isIntO ss3 = isIntO ss4 by simp

show react (oor Δ3 Δ1) ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

  show match1 (oor Δ3 Δ1) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor Δ3 Δ1) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 (oor Δ3 Δ1) ss3 ss4 statA ss1 ss2 statO
  proof(rule match12-simpleI, rule disjI1, intro conjI)
    fix ss3' ss4' statA'
    assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

  show ¬ isSecO ss3
  using v sa Δ3 unfolding ss by (simp add: Δ3-defs)

  show ¬ isSecO ss4
  using v sa Δ3 unfolding ss by (simp add: Δ3-defs)

  show stat: statA = statA' ∨ statO = Diff
  using v sa Δ3
  apply (cases ss3, cases ss4, cases ss1, cases ss2)
  apply (cases ss3', cases ss4', clarsimp)
  using v sa Δ3 unfolding ss statA' apply clarsimp
  apply(simp-all add: Δ3-defs sstatA'-def)
  apply(cases statO, simp-all) apply(cases statA, simp-all)
  unfolding finalS-defs
  by (smt (z3) Zero-neq-Suc list.size(3)
      map-eq-imp-length-eq status.exhaust newStat.simps)

have fence:prog ! pcOf (last cfs3) = Fence prog ! pcOf (last cfs4) = Fence
  using Δ3 unfolding ss apply-by(frul Δ3-implies, simp)+

  show oor Δ3 Δ1 ∞ ss3' ss4' statA' ss1 ss2 statO
  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-normal

```

```

    then show ?thesis using sa stat  $\Delta 3$  lcfgs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
  next
    case spec-mispred
    then show ?thesis by (simp add: fence)
  next
    case spec-normal
    then show ?thesis by (simp add: fence)
  next
    case spec-Fence note sf3 = spec-Fence
    have r4:  $\neg$  resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sa stat  $\Delta 3$ 
lcfgs sf3 unfolding ss by (simp add:  $\Delta 3$ -defs)
    show ?thesis
    using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-resolve
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-Fence note sf4 = spec-Fence
      show ?thesis
      apply(intro oorI2)
      using sa stat  $\Delta 3$  lcfgs v3 v4 sf3 sf4 unfolding ss hh
      apply(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
    qed (simp-all add: fence)
  next
    case spec-resolve note sr3 = spec-resolve
    show ?thesis
    using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next

```

```

      case spec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-normal
      then show ?thesis using sa stat  $\Delta 3$  lcfs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa stat  $\Delta 3$  lcfs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-resolve note sr4 = spec-resolve
      show ?thesis
      apply(intro oorI2)
      using sa stat  $\Delta 3$  lcfs v3 v4 sr3 sr4 unfolding ss hh
      apply(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
    qed (simp-all add: fence)
  qed (simp-all add: fence)
qed
qed
qed

```

lemma *stepe: unwindIntoCond* $\Delta 4$ $\Delta 4$

proof(rule *unwindIntoCond-simpleI*)

fix n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

assume r : *reachO* $ss3$ *reachO* $ss4$ *reachV* $ss1$ *reachV* $ss2$

and $\Delta 4$: $\Delta 4$ n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by (*cases* $ss3$, *auto*)

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by (*cases* $ss4$, *auto*)

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (*cases* $ss1$, *auto*)

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (*cases* $ss2$, *auto*)

note $ss = ss3$ $ss4$ $ss1$ $ss2$

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$cfg3$: $cfg3 = Config$ $pc3$ (*State* (*Vstore* $vs3$) $avst3$ $h3$ $p3$)

by (*cases* $cfg3$) (*metis* *state.collapse* *vstore.collapse*)

obtain $pc4$ $vs4$ $avst4$ $h4$ $p4$ **where**

$cfg4$: $cfg4 = Config$ $pc4$ (*State* (*Vstore* $vs4$) $avst4$ $h4$ $p4$)

by (*cases* $cfg4$) (*metis* *state.collapse* *vstore.collapse*)

```

note  $cfg = cfg3\ cfg4$ 

obtain  $hh3$  where  $h3: h3 = Heap\ hh3$  by( $cases\ h3, auto$ )
obtain  $hh4$  where  $h4: h4 = Heap\ hh4$  by( $cases\ h4, auto$ )
note  $hh = h3\ h4$ 

show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
  using  $\Delta_4\ Opt.final-def\ Prog.endPC-def\ finalS-def\ stepS-endPC$ 
  unfolding  $\Delta_4-defs\ ss$  apply  $clarify$ 
  by ( $metis\ Prog.finalN-defs(1)\ Prog.finalN-endPC\ Prog-axioms\ stepS-endPC$ )

then show  $isIntO\ ss3 = isIntO\ ss4$  by  $simp$ 

show  $react\ \Delta_4\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $react-def$  proof( $intro\ conjI$ )

  show  $match1\ \Delta_4\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match1-def$  by ( $simp\ add: finalS-def\ final-def$ )
  show  $match2\ \Delta_4\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match2-def$  by ( $simp\ add: finalS-def\ final-def$ )
  show  $match12\ \Delta_4\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  apply( $rule\ match12-simpleI$ ) using  $\Delta_4$  unfolding  $ss$  apply ( $simp\ add: \Delta_4-defs$ )
  by ( $simp\ add: stepS-endPC$ )
qed
qed

```

lemmas $theConds = step0\ step1\ step2\ step3\ stepe$

proposition $rsecure$

proof–

```

define  $m$  where  $m: m \equiv (5::nat)$ 
define  $\Delta s$  where  $\Delta s: \Delta s \equiv \lambda i::nat.$ 
   $if\ i = 0\ then\ \Delta 0$ 
   $else\ if\ i = 1\ then\ \Delta 1$ 
   $else\ if\ i = 2\ then\ \Delta 2$ 
   $else\ if\ i = 3\ then\ \Delta 3$ 
   $else\ \Delta 4$ 
define  $nxt$  where  $nxt: nxt \equiv \lambda i::nat.$ 
   $if\ i = 0\ then\ \{0,1::nat\}$ 
   $else\ if\ i = 1\ then\ \{1,2,3,4\}$ 
   $else\ if\ i = 2\ then\ \{1\}$ 
   $else\ if\ i = 3\ then\ \{3,1\}$ 
   $else\ \{4\}$ 
show  $?thesis$  apply( $rule\ distrib-unwind-rsecure[of\ m\ nxt\ \Delta s]$ )
  subgoal unfolding  $m$  by  $auto$ 
  subgoal unfolding  $nxt\ m$  by  $auto$ 
  subgoal using  $init$  unfolding  $\Delta s$  by  $auto$ 

```

```

    subgoal
      unfolding m next Δs apply (simp split: if-splits)
      using theConds
      unfolding oor-def oor3-def oor4-def by auto .
qed
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 (⟦⊥⟧)
```

```
consts NN::nat
```

```
lemma NN:int NN ≥ 0 by auto
```

```
consts size-aa1 :: nat
```

```
consts size-aa2 :: nat
```

```
consts mispred :: predState ⇒ pcounter list ⇒ bool
```

```
consts update :: predState ⇒ pcounter list ⇒ 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
```

```
unfolding vvars-defs by auto
```

```
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 $prog \equiv$

```
[
  // 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 $cases-7: (i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee i = 6 \vee i = 7 \vee i > 7$

```
apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
.....
```

lemma $xx-NN-cases: vs\ xx < int\ NN \vee vs\ xx \geq int\ NN$ **by** *auto*

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

```
pcOf cfg < 8  $\implies$  is-IfJump (prog ! (pcOf cfg))  $\longleftrightarrow$  pcOf cfg = 3
apply(cases cfg) subgoal for pc s using cases-7[of pcOf cfg ]
by (auto simp: prog-def) .
```

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

```
pc < 8  $\implies$  is-IfJump (prog ! pc)  $\longleftrightarrow$  pc = 3
using cases-7[of pc]
by (auto simp: prog-def)
```

lemma $eq-Fence-pc[simp]:$

```
pc < 8  $\implies$  prog ! pc = Fence  $\longleftrightarrow$  pc = 5
using cases-7[of pc]
by (auto simp: prog-def)
```

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*

by (*standard*, *simp add: prog-def*)

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** $\langle \rightarrow M \rangle$ 55)

where $x \rightarrow M y == \text{stepM } x y$

abbreviation

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

where $x \rightarrow N y == \text{stepN } x y$

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 $x \rightarrow N^* y == \text{star stepN } x y$

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

unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getTrustedInput-pcOf[simp]*: $pcOf\ cfg < 8 \implies is_getInput\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 1$

using *cases-7[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *getUntrustedInput-pcOf[simp]*: $prog!1 = Input\ U\ xx$
by (*auto simp: prog-def*)

lemma *getInput-not3[simp]*: $\neg is_getInput\ (prog!\ 3)$
by (*auto simp: prog-def*)

lemma *getInput-not4[simp]*: $\neg is_getInput\ (prog!\ 4)$
by (*auto simp: prog-def*)

lemma *Output-not4[simp]*: $\neg is_Output\ (prog!\ 4)$
by (*auto simp: prog-def*)

lemma *is-Output-pcOf[simp]*: $pcOf\ cfg < 8 \implies is_Output\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 7$

using *cases-7[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *is-Output*: $is_Output\ (prog!\ 7)$
unfolding *is-Output-def* **prog-def** **by** *auto*

lemma *is-Fence[simp]*: $(prog!\ 5) = Fence$
unfolding *prog-def* **by** *auto*

lemma *not-is-getTrustedInput[simp]*: $cfg = Config\ 3\ (State\ (Vstore\ vs)\ (Astore\ as)\ (Heap\ h)\ p) \implies \neg is_getInput\ (prog!\ pcOf\ cfg)$

unfolding *is-getInput-def* **prog-def** **by** *simp*

lemma *not-is-Output[simp]*: $cfg = Config\ pc\ (State\ (Vstore\ vs)\ (Astore\ as)\ (Heap\ h)\ p) \implies$

$pc = 3 \implies \neg is_Output\ (prog!\ pcOf\ cfg)$

unfolding *is-Output* **prog-def** **by** *simp*

lemma *isSecV-pcOf[simp]*:
 $isSecV\ (cfg, ibT, ibUT) \longleftrightarrow pcOf\ cfg = 0$
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf[simp]*:
 $isSecO\ (pstate, cfg, cfs, ibT, ibUT, ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
using *isSecO-def* **by** *simp*

lemma *getInputT-not[simp]*: $pcOf\ cfg < 8 \implies$
 $(prog\ !\ pcOf\ cfg) \neq Input\ T\ inp$
apply(*cases cfg*) **subgoal for** $pc\ s$ **using** *cases-7[of pcOf cfg]*]
by (*auto simp: prog-def*) .

lemma *getActV-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getActV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1\ then\ lhd\ ibUT\ else\ \perp)$
apply(*subst getActV-simps*) **unfolding** *prog-def*
apply *simp*
using *cases-7[of pcOf cfg]* **apply**(*elim disjE*)
using *getActV-simps not-is-getTrustedInput-getActV* **by** *auto*

lemma *getObsV-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getObsV\ (cfg, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 7\ then$
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $else\ \perp$
 $)$
apply(*subst getObsV-simps*)
unfolding *prog-def* **apply** *simp*
using *getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def* **by** *presburger*

lemma *getActO-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getActO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 $(if\ pcOf\ cfg = 1 \wedge cfgs = []\ then\ lhd\ ibUT\ else\ \perp)$
apply(*subst getActO-simps*)
apply(*cases cfgs, auto*)
unfolding *prog-def* **apply** *simp*
using *getActV-simps getActV-pcOf prog-def* **by** *presburger*

lemma *getObsO-pcOf[simp]*:
 $pcOf\ cfg < 8 \implies$
 $getObsO\ (pstate, cfg, cfgs, ibT, ibUT, ls) =$
 $(if\ (pcOf\ cfg = 7 \wedge cfgs = [])\ then$
 $(outOf\ (prog!(pcOf\ cfg))\ (stateOf\ cfg),\ ls)$
 $else\ \perp$
 $)$
apply(*subst getObsO-simps*)
apply(*cases cfgs, auto*)
unfolding *prog-def* **apply** *simp*
using *getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def* **by** *presburger*

lemma *eqSec-pcOf[simp]*:
 $eqSec\ (cfg1, ibT, ibUT1, ls1)\ (pstate3, cfg3, cfgs3, ibT, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfgs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
unfolding *eqSec-def* **by** *simp*

lemma *nextB-pc0[simp]*:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc0[simp]*:
 $readLocs\ (Config\ 0\ s) = \{\}$
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1[simp]*:
 $ibUT \neq LNil \implies nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p), ibT, ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1[simp]*:
 $readLocs\ (Config\ 1\ s) = \{\}$
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1'[simp]*:
 $ibUT \neq LNil \implies nextB\ (Config\ (Suc\ 0)\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p), ibT, ltl\ ibUT)$
apply(*subst nextB-getUntrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1'[simp]*:
 $readLocs\ (Config\ (Suc\ 0)\ s) = \{\}$
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc2[simp]*:
readLocs (*Config 2* *s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
by(*cases* *vs* *xx* < *NN*, *auto*)

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*)
apply(*subst nextM-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextM-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
by(*cases* *vs* *xx* < *NN*, *auto*)

lemma *readLocs-pc3[simp]*:

$readLocs (Config\ 3\ s) = \{\}$
unfolding *endPC-def readLocs-def unfolding prog-def by auto*

lemma *nextB-pc4[simp]*:
 $nextB (Config\ 4 (State (Vstore\ vs)\ avst (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $(let\ l = array-loc\ aa1 (nat (vs\ xx))\ avst$
 $in (Config\ 5 (State (Vstore (vs(vv := h\ l))))\ avst (Heap\ h)\ p)),\ ibT,\ ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def unfolding prog-def by auto*

lemma *readLocs-pc4[simp]*:
 $readLocs (Config\ 4 (State (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa1 (nat (vs\ xx))\ avst\}$
unfolding *endPC-def readLocs-def unfolding prog-def by auto*

lemma *nextB-pc5[simp]*:
 $nextB (Config\ 5\ s,\ ibT,\ ibUT) = (Config\ 6\ s,\ ibT,\ ibUT)$
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def unfolding prog-def by auto*

lemma *readLocs-pc5[simp]*:
 $readLocs (Config\ 5\ s) = \{\}$
unfolding *endPC-def readLocs-def unfolding prog-def by auto*

lemma *nextB-pc6[simp]*:
 $nextB (Config\ 6 (State (Vstore\ vs)\ avst (Heap\ h)\ p),\ ibT,\ ibUT) =$
 $(let\ l = array-loc\ aa2 (nat (vs\ vv * 512))\ avst$
 $in (Config\ 7 (State (Vstore (vs(tt := h\ l))))\ avst (Heap\ h)\ p)),\ ibT,\ ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def unfolding prog-def by auto*

lemma *readLocs-pc6[simp]*:
 $readLocs (Config\ 6 (State (Vstore\ vs)\ avst\ h\ p)) = \{array-loc\ aa2 (nat (vs\ vv * 512))\ avst\}$
unfolding *endPC-def readLocs-def unfolding prog-def by auto*

lemma *nextB-pc7[simp]*:
 $nextB (Config\ 7\ s,\ ibT,\ ibUT) = (Config\ 8\ s,\ ibT,\ ibUT)$
apply(*subst nextB-Output*)

unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc7[simp]*:

readLocs (*Config* 7 *s*) = {}

unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-stepB-pc*:

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

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

apply(*cases* *s*) **subgoal for** *vst avst hh p* **apply**(*cases* *vst, cases avst, cases hh*)

subgoal for *vs as h*

using *cases-7[of pc]* **apply** *safe*

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def, metis llist.collapse*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply(*cases* *vs xx < NN*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*) .

subgoal apply(*cases* *vs xx < NN*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*) .

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply *simp* **apply**(*subst* *stepB.simps*) **unfolding** *endPC-def*
by (*simp* *add: prog-def*)

subgoal apply simp apply(subst stepB.simps) **unfolding** endPC-def
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding** endPC-def
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding** endPC-def
by (simp add: prog-def)

subgoal by auto

subgoal by auto

...

lemma not-finalB:

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

$\neg finalB (Config\ pc\ s, ibT, ibUT)$

using nextB-stepB-pc **by** (simp add: stepB-iff-nextB)

lemma finalB-pc-iff':

$pc < 8 \implies$

$finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$

$(pc = 1 \wedge ibUT = LNil)$

subgoal apply safe

subgoal using nextB-stepB-pc[of pc] **by** (auto simp add: stepB-iff-nextB)

subgoal using nextB-stepB-pc[of pc] **by** (auto simp add: stepB-iff-nextB)

subgoal using finalB-iff getUntrustedInput-pcOf **by** auto . .

lemma finalB-pc-iff:

$pc \leq 8 \implies$

$finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$

$(pc = 1 \wedge ibUT = LNil \vee pc = 8)$

using cases-7[of pc] **apply** (elim disjE, simp add: finalB-def)

subgoal by (meson final-def stebB-0)

by (simp add: finalB-pc-iff' finalB-endPC)+

lemma finalB-pcOf-iff[simp]:

$pcOf\ cfg \leq 8 \implies$

$finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = 8)$

by (metis config.collapse finalB-pc-iff)

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

apply(rule notI, cases cfg)

subgoal for pc s **apply**(cases s)

subgoal for vst avst hh p **apply**(cases vst, cases avst, cases hh)

subgoal for vs as h

using cases-7[of pc] **apply**(elim disjE) **unfolding** finalS-defs

subgoal by(erule allE[of - (pstate, Config 1 (State (Vstore vs) avst hh p), []),

$ibT, ibUT, ls]$, *erule notE*, *rule nonspec-normal*, *auto*)

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*

$pstate\ pstate\ ibT\ ibUT$
Config 2 (State (Vstore (vs(xx:= lhd ibUT))) avst hh

$p)$

$ibT\ ltl\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
 $vs)\ avst\ hh\ p))\ ls]$

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*

$pstate\ pstate\ ibT\ ibUT$
Config 3 (State (Vstore (vs(tt:= 0))) avst hh p)
 $ibT\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
 $vs)\ avst\ hh\ p))\ ls]$

prefer 7 subgoal by metis by simp-all

subgoal apply(*cases mispred pstate [3]*)

subgoal by(*erule allE*[of - (*update pstate [pcOf (Config pc (State (Vstore vs) avst hh p)]*),

Config (if vs xx < NN then 4 else 7) (State (Vstore

$vs)\ avst\ hh\ p)$,

$[Config\ (if\ vs\ xx\ <\ NN\ then\ 7\ else\ 4)\ (State\ (Vstore\ vs)$
 $avst\ hh\ p)]$,

$ibT, ibUT, ls]$, *erule notE*, *rule nonspec-mispred*, *auto*
simp: finalM-iff)

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*

$pstate\ pstate\ ibT\ ibUT$
Config (if vs xx < NN then 4 else 7) (State (Vstore

$vs)\ avst\ hh\ p)$

$ibT\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
 $vs)\ avst\ hh\ p))\ ls]$

prefer 7 subgoal by metis apply simp-all by (simp add: nextB-pc3)

.

subgoal by(*erule allE*[of - (*pstate, Config 5 (State (Vstore (vs(vv := h (array-loc aa1 (nat (vs xx)) avst)))) avst hh p)*,

$[],\ ibT, ibUT, ls\ \cup\ \{array-loc$
 $aa1\ (nat\ (vs\ xx))\ avst\ \}$]),

erule notE, *rule nonspec-normal*, *auto*)

subgoal by(*erule allE*[of - (*pstate, Config 6 (State (Vstore vs) avst hh p)*, $[],$

$ibT, ibUT, ls]$, *erule notE*, *rule nonspec-normal*, *auto*)

subgoal by(*erule allE*[of - (*pstate, Config 7 (State (Vstore (vs(tt := h (array-loc aa2 (nat (vs vv * 512)) (Avstore as)))) avst hh p)*,

[], *ibT,ibUT, ls* \cup {*array-loc aa2* (*nat (vs vv * 512)*)}

(*Avstore as*)]],
erule notE, rule nonspec-normal, auto)

subgoal by(*erule allE*[*of - (pstate, Config 8 (State (Vstore vs) avst hh p)*, [],
ibT,ibUT, ls)], *erule notE, rule nonspec-normal, auto*)

by *simp-all . . .*

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)*

using *not-is-getTrustedInput not-is-Output*

apply(*cases cfg*)

subgoal for *pc s* **apply**(*cases s*)

subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)

subgoal for *vs as h* **apply**(*cases last cfgs*)

subgoal for *pcs ss* **apply**(*cases ss*)

subgoal for *vsts avsts hhs ps* **apply**(*cases vsts, cases avsts, cases hhs, simp*)

apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)

subgoal unfolding *finalS-defs* **using** *spec-resolve* **by** (*metis list.size(3) n-not-Suc-n*)

subgoal for *vss ass hs* **apply**(*elim disjE, elim conjE, elim disjE, simp*)

unfolding *finalS-defs*

subgoal apply(*rule notI,*

erule allE[*of - (pstate, Config 7 (State (Vstore vs) (Avstore as) (Heap h) p),*

[*Config 5 (State (Vstore (vss(vv := hs (array-loc aa1 (nat (vss*

xx) avsts)))) avsts hhs ps]],

ibT,ibUT,ls \cup *readLocs (last cfgs)*]])

by(*erule notE,*

rule spec-normal[*of - - - - Config 5 (State (Vstore (vss(vv := hs (array-loc*
aa1 (nat (vss xx) avsts)))) avsts hhs ps]], *auto*)

subgoal apply(*rule notI,*

erule allE[*of - (pstate, Config 7 (State (Vstore vs) (Avstore as) (Heap h)*

p), [], *ibT,ibUT,ls*]])

using *spec-Fence* **by** (*metis Zero-not-Suc config.sel(1) is-Fence length-0-conv*)

subgoal apply(*rule notI,*

erule allE[*of - (update pstate (4 # map pcOf cfgs), Config 4 (State (Vstore vs)*
(Avstore as) (Heap h) p),

[], *ibT,ibUT,ls*]])

using *spec-resolve*

by (*metis config.sel(1) is-Output length-1-butlast length-greater-0-conv*
less-Suc0)

.....
end

10.2 Proof

theory *Fun3-secure*
imports *Fun3*
begin

type-synonym *stateO* = *configS*
type-synonym *stateV* = *config* × *val llist* × *val llist* × *loc set*

definition *PC* ≡ {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* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status*
⇒ *bool*

where

common = (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* ∧
cfg1 = *cfg3* ∧ *cfg2* = *cfg4* ∧
pcOf *cfg3* = *pcOf* *cfg4* ∧ *map pcOf* *cfgs3* = *map pcOf* *cfgs4* ∧
pcOf *cfg3* ∈ *PC* ∧ *pcOf* ' (*set* *cfgs3*) ⊆ *PC* ∧
///
array-base aa1 (*getAvstore* (*stateOf* *cfg3*)) = *array-base aa1* (*getAvstore* (*stateOf*
cfg4)) ∧
(∀ *cfg3'* ∈ *set* *cfgs3*. *array-base aa1* (*getAvstore* (*stateOf* *cfg3'*)) = *array-base aa1*
(*getAvstore* (*stateOf* *cfg3*))) ∧
(∀ *cfg4'* ∈ *set* *cfgs4*. *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* *cfgs3*. *array-base aa2* (*getAvstore* (*stateOf* *cfg3'*)) = *array-base aa2*
(*getAvstore* (*stateOf* *cfg3*))) ∧

$(\forall \text{cfg4}' \in \text{set } \text{cfgs4}. \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg4}')) = \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg4}))) \wedge$
 $///$
 $(\text{statA} = \text{Diff} \longrightarrow \text{statO} = \text{Diff}))$

lemma *common-implies: common*

$(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO} \implies$
 $\text{pcOf } \text{cfg1} < 9 \wedge \text{pcOf } \text{cfg2} = \text{pcOf } \text{cfg1}$
unfolding *common-def PC-def* **by** (*auto simp: image-def subset-eq*)

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**

$\Delta 0 = (\lambda \text{num}$
 $(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO}.$
 $(\text{common } (\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO} \wedge$
 $\text{ibUT1} = \text{ibUT3} \wedge \text{ibUT2} = \text{ibUT4} \wedge$
 $(\text{pcOf } \text{cfg3} > 1 \longrightarrow \text{same-var-o } \text{xx } \text{cfg3 } \text{cfgs3 } \text{cfg4 } \text{cfgs4}) \wedge$
 $(\text{pcOf } \text{cfg3} < 2 \longrightarrow \text{ibUT1} \neq \text{LNil} \wedge \text{ibUT2} \neq \text{LNil} \wedge \text{ibUT3} \neq \text{LNil} \wedge \text{ibUT4} \neq \text{LNil})$
 \wedge
 $\text{pcOf } \text{cfg3} \in \text{beforeInput} \wedge$
 $\text{ls1} = \text{ls3} \wedge \text{ls2} = \text{ls4} \wedge$
 $\text{noMisSpec } \text{cfgs3}$
 $))$

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

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

$(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$

$(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $(pcOf\ cfg3 = 1 \longrightarrow ibUT3 \neq LNil) \wedge$
 $(pcOf\ cfg4 = 1 \longrightarrow ibUT4 \neq LNil) \wedge$
 $pcOf\ cfg1 < 8 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $cfgs3 = [] \wedge pcOf\ cfg3 < 8 \wedge$
 $cfgs4 = [] \wedge pcOf\ cfg4 < 8$
unfolding $\Delta 0$ -defs
apply (intro conjI)
apply simp-all
by (metis map-is-Nil-conv)

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

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

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

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

$(pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg1 < 8 \wedge$
 $cfgs3 = [] \wedge pcOf\ cfg3 \neq 1 \wedge pcOf\ cfg3 < 8 \wedge$
 $cfgs4 = [] \wedge pcOf\ cfg4 \neq 1 \wedge pcOf\ cfg4 < 8$
unfolding $\Delta 1$ -defs

apply(*intro conjI*) **apply** *simp-all*
using *One-nat-def verit-eq-simplify(10,12)* **apply** *linarith*
apply (*metis list.map-disc-iff*)
by *linarith*

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, cfigs3, ibT3, ibUT3, ls3*)
 (*pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4*)
statA
 (*cfg1, ibT1, ibUT1, ls1*)
 (*cfg2, ibT2, ibUT2, ls2*)
statO.
 (*common*
 (*pstate3, cfg3, cfigs3, ibT3, ibUT3, ls3*)
 (*pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4*)
statA
 (*cfg1, ibT1, ibUT1, ls1*)
 (*cfg2, ibT2, ibUT2, ls2*)
statO \wedge
pcOf *cfg3* = *startOfThenBranch* \wedge
pcOf (*last cfigs3*) = *elseBranch* \wedge
same-var-o *xx* *cfg3* *cfigs3* *cfg4* *cfigs4* \wedge
ls1 = *ls3* \wedge *ls2* = *ls4* \wedge
misSpecL1 *cfigs3*
))

lemmas $\Delta 2\text{-defs} = \Delta 2\text{-def}$ *common-def* *PC-def* *same-var-def* *startOfThenBranch-def*

misSpecL1-def *elseBranch-def*

lemma $\Delta 2\text{-implies: } \Delta 2 \text{ num}$

(*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*) = 7 \wedge *pcOf* *cfg3* = 4 \wedge
pcOf (*last cfigs4*) = *pcOf* (*last cfigs3*) \wedge
pcOf *cfg3* = *pcOf* *cfg4* \wedge
length *cfigs3* = *Suc 0* \wedge
length *cfigs3* = *length* *cfigs4*
apply(*intro conjI*)
unfolding $\Delta 2\text{-defs}$ **apply** *simp-all*
apply (*simp add: image-subset-iff*)

apply (*metis last-map map-is-Nil-conv*)
by (*metis length-map*)

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*, *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
pcOf (*last* *cfgs3*) \in *inThenBranchBeforeFence* \wedge
same-var-o *xx* *cfg3* *cfgs3* *cfg4* *cfgs4* \wedge
Language-Prelims.dist *ls3* *ls4* \subseteq *Language-Prelims.dist* *ls1* *ls2* \wedge
(*pcOf* (*last* *cfgs3*) = 4 \longrightarrow *ls1* = *ls3* \wedge *ls2* = *ls4*) \wedge
misSpecL1 *cfgs3*
))

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

lemma $\Delta 3\text{-implies}$: $\Delta 3$ *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*

```

apply(intro conjI)
  unfolding  $\Delta 3$ -defs apply simp-all
  apply (simp add: image-subset-iff)
  apply (metis last-map map-is-Nil-conv)
  apply (metis last-in-set list.size(3) n-not-Suc-n)
  apply (metis One-nat-def last-in-set length-0-conv length-map zero-neq-one)
  apply (metis length-map)
  by (metis last-in-set list.map-disc-iff)

```

definition $\Delta 1' :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

```

 $\Delta 1' = (\lambda \text{num}$ 
  (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 \Longrightarrow
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
unfolding $\Delta 1'$ -*defs*

apply(*intro conjI*) **apply** *simp-all*
by (*metis list.map-disc-iff*)

definition $\Delta_4 :: \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status}$
 $\Rightarrow \text{bool}$ **where**

$\Delta_4 = (\lambda \text{num}$
 $(\text{pstate3}, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $(\text{pstate4}, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 statA
 $(\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 statO
 $(\text{pcOf } \text{cfg3} = \text{endPC} \wedge \text{pcOf } \text{cfg4} = \text{endPC} \wedge \text{cfgs3} = [] \wedge \text{cfgs4} = [] \wedge$
 $\text{pcOf } \text{cfg1} = \text{endPC} \wedge \text{pcOf } \text{cfg2} = \text{endPC}))$

lemmas $\Delta_4\text{-defs} = \Delta_4\text{-def}$ *common-def* *endPC-def*

lemma *init: initCond* Δ_0

unfolding *initCond-def* **apply**(*intro allI*)
subgoal for $s_3 s_4$ **apply**(*cases s3, cases s4*)
subgoal for $\text{pstate3 } \text{cfg3 } \text{cfgs3 } \text{ibT3 } \text{ibUT3 } \text{ls3 } \text{pstate4 } \text{cfg4 } \text{cfgs4 } \text{ibT4 } \text{ibUT4 } \text{ls4}$

apply *clarify*
apply(*rule exI[of - (cfg3, ibT3, ibUT3, ls3)]*)
apply(*cases getAvstore (stateOf cfg3)*)
apply(*rule exI[of - (cfg4, ibT4, ibUT4, ls4)]*)
apply(*cases getAvstore (stateOf cfg4)*)
unfolding $\Delta_0\text{-defs}$ *array-base-def* **by** *auto* . .

lemma *step0: unwindIntoCond* Δ_0 (*oor* Δ_0 Δ_1)

proof(*rule unwindIntoCond-simpleI*)

fix $n \text{ ss3 } \text{ss4 } \text{statA } \text{ss1 } \text{ss2 } \text{statO}$
assume $r: \text{reachO } \text{ss3 } \text{reachO } \text{ss4 } \text{reachV } \text{ss1 } \text{reachV } \text{ss2}$
and $\Delta_0: \Delta_0 n \text{ ss3 } \text{ss4 } \text{statA } \text{ss1 } \text{ss2 } \text{statO}$

obtain $\text{pstate3 } \text{cfg3 } \text{cfgs3 } \text{ibT3 } \text{ibUT3 } \text{ls3}$ **where** $\text{ss3}: \text{ss3} = (\text{pstate3}, \text{cfg3}, \text{cfgs3},$
 $\text{ibT3}, \text{ibUT3}, \text{ls3})$

by (*cases ss3, auto*)

obtain $\text{pstate4 } \text{cfg4 } \text{cfgs4 } \text{ibT4 } \text{ibUT4 } \text{ls4}$ **where** $\text{ss4}: \text{ss4} = (\text{pstate4}, \text{cfg4}, \text{cfgs4},$
 $\text{ibT4}, \text{ibUT4}, \text{ls4})$

by (*cases ss4, auto*)

obtain $\text{cfg1 } \text{ibT1 } \text{ibUT1 } \text{ls1}$ **where** $\text{ss1}: \text{ss1} = (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$

by (*cases ss1, auto*)

obtain $\text{cfg2 } \text{ibT2 } \text{ibUT2 } \text{ls2}$ **where** $\text{ss2}: \text{ss2} = (\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$

```

by (cases ss2, auto)
note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
note hh = h3 h4

have f1:¬finalN ss1
  using Δ0 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ0-defs
  by simp

have f2:¬finalN ss2
  using Δ0 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ0-defs
  by simp

have f3:¬finalS ss3
  using Δ0 unfolding ss apply-apply(frule Δ0-implies)
  using finalS-cond by simp

have f4:¬finalS ss4
  using Δ0 unfolding ss apply-apply(frule Δ0-implies)
  using finalS-cond by simp

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react (oor Δ0 Δ1) ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

  show match1 (oor Δ0 Δ1) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-defs)
  show match2 (oor Δ0 Δ1) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-defs)
  show match12 (oor Δ0 Δ1) ss3 ss4 statA ss1 ss2 statO

```

```

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain pc3 vs3 avst3 h3 p3 where
  cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
  cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  show eqSec ss1 ss3
  using v sa  $\Delta 0$  unfolding ss
  by (simp add:  $\Delta 0$ -defs eqSec-def)

  show eqSec ss2 ss4
  using v sa  $\Delta 0$  unfolding ss
  apply (simp add:  $\Delta 0$ -defs eqSec-def)
  by (metis length-0-conv length-map)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 0$  unfolding ss
  unfolding Opt.eqAct-def Van.eqAct-def
  apply(simp-all add:  $\Delta 0$ -defs)
  by (metis f3 map-is-Nil-conv ss3)

  show match12-12 (oor  $\Delta 0$   $\Delta 1$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
    show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff

```

```

show sstatO' statO ss1 ss2 = Diff
using v sa  $\Delta 0$  sstat unfolding ss cfg statA' apply simp
apply(simp add:  $\Delta 0$ -defs sstatO'-def sstatA'-def finalS-def final-def)
using cases-7[of pc3] apply(elim disjE)
apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
apply(cases statO, simp-all) apply (cases statA, simp-all)
by (smt (z3) status.distinct status.exhaust newStat.simps)+
} note stat = this
have cfs3:cfs3 = [] using  $\Delta 0$  unfolding ss by (simp add:  $\Delta 0$ -defs)

show oor  $\Delta 0$   $\Delta 1$   $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO' statO
ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case spec-normal
then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-mispred
then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-Fence
then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-resolve
then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case nonspec-mispred
then show ?thesis using sa  $\Delta 0$  stat unfolding ss apply (simp add:
 $\Delta 0$ -defs)
by (metis is-If-pc less-Suc-eq nat-less-le numeral-1-eq-Suc-0 nu-
meral-3-eq-3
one-eq-numeral-iff semiring-norm(83) zero-less-numeral zero-neq-numeral)

next
case nonspec-normal note nn3 = nonspec-normal
show ?thesis
using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case nonspec-mispred
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-normal
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next

```

```

      case spec-mispred
      then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
    next
      case spec-resolve
      then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
    next
      case nonspec-normal note nn4=nonspec-normal
      show ?thesis using sa stat  $\Delta 0$  v3 v4 nn3 nn4 f4 unfolding ss cfg hh
Opt.eqAct-def
      apply clarsimp
      using cases-7[of pc3] apply (elim disjE)
      subgoal apply (rule oorI1) by (simp add:  $\Delta 0$ -defs)
      subgoal apply (rule oorI2) apply (simp add:  $\Delta 0$ -defs, auto)
        unfolding  $\Delta 1$ -defs
        subgoal by (simp add:  $\Delta 0$ -defs)
        subgoal by (simp add:  $\Delta 0$ -defs) .
        by (simp add:  $\Delta 0$ -defs)+
      qed (simp-all add: cfgs3)
      qed (simp-all add: cfgs3)
    qed
  qed
  qed
  qed

```

```

lemma step1: unwindIntoCond  $\Delta 1$  (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ )
proof (rule unwindIntoCond-simpleI)
  fix n ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 1$ :  $\Delta 1$  n ss3 ss4 statA ss1 ss2 statO

```

```

  obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

```

```

obtain  $pc1\ vs1\ avst1\ h1\ p1$  where
 $cfg1: cfg1 = Config\ pc1\ (State\ (Vstore\ vs1)\ avst1\ h1\ p1)$ 
by ( $cases\ cfg1$ ) ( $metis\ state.collapse\ vstore.collapse$ )
obtain  $pc2\ vs2\ avst2\ h2\ p2$  where
 $cfg2: cfg2 = Config\ pc2\ (State\ (Vstore\ vs2)\ avst2\ h2\ p2)$ 
by ( $cases\ cfg2$ ) ( $metis\ state.collapse\ vstore.collapse$ )
obtain  $pc3\ vs3\ avst3\ h3\ p3$  where
 $cfg3: cfg3 = Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)$ 
by ( $cases\ cfg3$ ) ( $metis\ state.collapse\ vstore.collapse$ )
obtain  $pc4\ vs4\ avst4\ h4\ p4$  where
 $cfg4: cfg4 = Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)$ 
by ( $cases\ cfg4$ ) ( $metis\ state.collapse\ vstore.collapse$ )
note  $cfg = cfg1\ cfg2\ cfg3\ cfg4$ 

obtain  $hh3$  where  $h3: h3 = Heap\ hh3$  by( $cases\ h3, auto$ )
obtain  $hh4$  where  $h4: h4 = Heap\ hh4$  by( $cases\ h4, auto$ )
note  $hh = h3\ h4$ 

have  $f1: \neg finalN\ ss1$ 
using  $\Delta1\ finalB-pc-iff'$  unfolding  $ss\ cfg\ finalN-iff-finalB\ \Delta1-defs$ 
by  $simp\ linarith$ 

have  $f2: \neg finalN\ ss2$ 
using  $\Delta1\ finalB-pc-iff'$  unfolding  $ss\ cfg\ finalN-iff-finalB\ \Delta1-defs$ 
by  $simp\ linarith$ 

have  $f3: \neg finalS\ ss3$ 
using  $\Delta1\ unfolding\ ss\ apply-apply(frule\ \Delta1-implies)$ 
using  $finalS-cond$  by  $simp$ 

have  $f4: \neg finalS\ ss4$ 
using  $\Delta1\ unfolding\ ss\ apply-apply(frule\ \Delta1-implies)$ 
using  $finalS-cond$  by  $simp$ 

note  $finals = f1\ f2\ f3\ f4$ 

show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
using  $finals$  by  $auto$ 

then show  $isIntO\ ss3 = isIntO\ ss4$  by  $simp$ 

show  $react\ (oor4\ \Delta1\ \Delta2\ \Delta3\ \Delta4)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $react-def$  proof( $intro\ conjI$ )

show  $match1\ (oor4\ \Delta1\ \Delta2\ \Delta3\ \Delta4)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $match1-def$  by ( $simp\ add: finalS-def\ final-def$ )
show  $match2\ (oor4\ \Delta1\ \Delta2\ \Delta3\ \Delta4)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $match2-def$  by ( $simp\ add: finalS-def\ final-def$ )

```

```

show match12 (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ ) ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  show eqSec ss1 ss3
  using v sa  $\Delta 1$  unfolding ss
  by (simp add:  $\Delta 1$ -defs eqSec-def)

  show eqSec ss2 ss4
  using v sa  $\Delta 1$  unfolding ss
  apply (simp add:  $\Delta 1$ -defs eqSec-def)
  by (metis length-0-conv length-map)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 1$  unfolding ss Van.eqAct-def
  apply (simp-all add:  $\Delta 1$ -defs)
  by linarith

  show match12-12 (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
  conjI impI)
    show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
  using v sa  $\Delta 1$  sstat unfolding ss cfg statA'
  apply(simp add:  $\Delta 1$ -defs sstatO'-def sstatA'-def)
  using cases-7[of pc3] apply(elim disjE)
  defer 1 defer 1
  subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  }
```

```

      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    by simp+
  } note stat = this
  have cfs:cfgs3 = [] cfs4 = [] using  $\Delta 1$ -implies[OF  $\Delta 1$ [unfolded ss]] by
auto

  show (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta 4$ )  $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2)
(ssstatO' statO ss1 ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case spec-normal
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-mispred
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-Fence
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-resolve
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case nonspec-mispred note nm3 = nonspec-mispred
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

    case nonspec-normal
  then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-normal
  then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-mispred
  then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:

```

```

Δ1-defs)
  next
    case spec-Fence
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-resolve
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case nonspec-mispred note nm4 = nonspec-mispred
    then show ?thesis
    using sa Δ1 stat v3 v4 nm3 nm4 unfolding ss cfg hh apply clarsimp
    using cases-7[of pc3] apply(elim disjE)
    subgoal by simp
    subgoal by simp
    subgoal by simp
    subgoal using xx-NN-cases[of vs3] apply(elim disjE)
    subgoal apply(rule oor4I2) by (simp add: Δ1-defs Δ2-defs)
    subgoal apply(rule oor4I3) by (simp add: Δ1-defs Δ3-defs) .
    by (simp-all add: Δ1-defs)+
    qed(simp-all add: cfgs)
  next
    case nonspec-normal note nn3 = nonspec-normal
    show ?thesis using v4 [unfolded ss, simplified] proof (cases rule: stepS-cases)

      case nonspec-mispred
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-normal
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-mispred
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-Fence
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-resolve
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case nonspec-normal
      then show ?thesis using sa Δ1 stat v3 v4 nn3 unfolding ss cfg hh
apply clarsimp

```

```

using cases-7[of pc3] apply(elim disjE)
  subgoal by (simp add: Δ1-defs)
  subgoal by (simp add: Δ1-defs)
  subgoal apply(rule oor4I1) by(simp add:Δ1-defs)
  subgoal using xx-NN-cases[of vs3] apply(elim disjE)
    subgoal apply(rule oor4I1) by (simp add: Δ1-defs)
    subgoal apply(rule oor4I1) by (simp add: Δ1-defs) .
  subgoal apply(rule oor4I1) by (simp add: Δ1-defs)
  subgoal apply(rule oor4I1) by (simp add: Δ1-defs)
  subgoal apply(rule oor4I1) by (simp add: Δ1-defs)
  subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δ4-defs)
  subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δ4-defs) .
qed(simp-all add: cfgs)
qed(simp-all add: cfgs)
qed
qed
qed
qed

```

lemma step2: unwindIntoCond Δ2 Δ1

proof(rule unwindIntoCond-simpleI)

fix n ss3 ss4 statA ss1 ss2 statO

assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2

and Δ2: Δ2 n ss3 ss4 statA ss1 ss2 statO

obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 **where** ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)

by (cases ss3, auto)

obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 **where** ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)

by (cases ss4, auto)

obtain cfg1 ibT1 ibUT1 ls1 **where** ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)

by (cases ss1, auto)

obtain cfg2 ibT2 ibUT2 ls2 **where** ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)

by (cases ss2, auto)

note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 **where**

lcfgs3: last cfgs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)

by (cases last cfgs3) (metis state.collapse vstore.collapse)

obtain pc4 vs4 avst4 h4 p4 **where**

lcfgs4: last cfgs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)

by (cases last cfgs4) (metis state.collapse vstore.collapse)

note lcfgs = lcfgs3 lcfgs4

have f1: ¬finalN ss1

using Δ2 finalB-pc-iff' **unfolding** ss finalN-iff-finalB Δ2-defs

```

by auto

have f2:¬finalN ss2
  using Δ2 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ2-defs
  by auto

have f3:¬finalS ss3
  using Δ2 unfolding ss apply-apply(frule Δ2-implies)
  using finalS-cond-spec by simp

have f4:¬finalS ss4
  using Δ2 unfolding ss apply-apply(frule Δ2-implies)
  using finalS-cond-spec by simp

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react Δ1 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 Δ1 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 Δ1 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 Δ1 ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI1, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
  note hh = h3 h4

```

```

show  $\neg$  isSecO ss3
using v sa  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

show  $\neg$  isSecO ss4
using v sa  $\Delta 2$  unfolding ss apply clarsimp
by (simp add:  $\Delta 2$ -defs, linarith)

show stat: statA = statA'  $\vee$  statO = Diff
using v sa  $\Delta 2$ 
apply (cases ss3, cases ss4, cases ss1, cases ss2)
apply (cases ss3', cases ss4', clarsimp)
unfolding ss statA' apply clarsimp
apply (simp-all add:  $\Delta 2$ -defs sstatA'-def)
apply (cases statO, simp-all) apply (cases statA, simp-all)
unfolding finalS-defs
by (smt (verit, ccfv-SIG) newStat.simps(1))

  have isO:is-Output (prog ! pcOf (last cfigs3)) is-Output (prog ! pcOf (last
cfigs4)) using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by (simp add: is-Output)+
  have pstate3:pstate3 = pstate4 using  $\Delta 2$ [unfolded ss  $\Delta 2$ -defs] by fast
  show  $\Delta 1 \infty$  ss3' ss4' statA' ss1 ss2 statO

using v3[unfolded ss, simplified] proof (cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case nonspec-mispred
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case spec-normal
  then show ?thesis using isO by auto
next
  case spec-mispred
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply-
  apply (frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
next
  case spec-Fence
  then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply-
  apply (frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolveI
  then show ?thesis using isO by blast
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfigs4) cfigs4  $\neq$  []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis using v4[unfolded ss, simplified] proof (cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis by (simp add: r4)

```

```

next
  case nonspec-mispred
  then show ?thesis by (simp add: r4)
next
  case spec-normal
  then show ?thesis by (simp add: isO)
next
  case spec-mispred
  then show ?thesis using r4 by auto
next
  case spec-Fence
  then show ?thesis using r4 by auto
next
  case spec-resolveI
  then show ?thesis using isO by blast
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply (frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs) by clarsimp
next
  case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
  show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply (frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs) by clarsimp
qed
next
  case spec-resolveO note sr3 = spec-resolveO
  then have cfgs4: cfgs4  $\neq$  [] using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by auto
  show ?thesis using v4[unfolded ss, simplified] proof (cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis by (simp add: cfgs4)
  next
    case nonspec-mispred
    then show ?thesis by (simp add: cfgs4)
  next
    case spec-normal
    then show ?thesis by (simp add: isO)
  next
    case spec-mispred
    then show ?thesis using isO by auto
  next
    case spec-Fence
    then show ?thesis using isO by auto
  next
    case spec-resolveI
    then show ?thesis using isO by blast
  next
    case spec-resolve note sr4 = spec-resolve

```

```

show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
unfolding ss lcfgs hh apply-
apply(frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs) by clarsimp
next
case spec-resolveO note sr4 = spec-resolveO(1,3-) isO(2)
show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
unfolding ss lcfgs hh apply-
apply(frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs) by clarsimp
qed
qed
qed
qed

```

lemma step3: unwindIntoCond $\Delta 3$ (oor $\Delta 3$ $\Delta 1'$)

proof(rule unwindIntoCond-simpleI)

fix n ss3 ss4 statA ss1 ss2 statO

assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2

and $\Delta 3$: $\Delta 3$ n ss3 ss4 statA ss1 ss2 statO

obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 **where** ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)

by (cases ss3, auto)

obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 **where** ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)

by (cases ss4, auto)

obtain cfg1 ibT1 ibUT1 ls1 **where** ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)

by (cases ss1, auto)

obtain cfg2 ibT2 ibUT2 ls2 **where** ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)

by (cases ss2, auto)

note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 **where**

lcfgs3: last cfgs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)

by (cases last cfgs3) (metis state.collapse vstore.collapse)

obtain pc4 vs4 avst4 h4 p4 **where**

lcfgs4: last cfgs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)

by (cases last cfgs4) (metis state.collapse vstore.collapse)

note lcfgs = lcfgs3 lcfgs4

obtain hh3 **where** h3: h3 = Heap hh3 **by**(cases h3, auto)

obtain hh4 **where** h4: h4 = Heap hh4 **by**(cases h4, auto)

note hh = h3 h4

have f1: \neg finalN ss1

using $\Delta 3$ finalB-pc-iff' **unfolding** ss finalN-iff-finalB $\Delta 3$ -defs

by auto

```

have f2:¬finalN ss2
  using Δ3 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ3-defs
  by auto

have f3:¬finalS ss3
  using Δ3 unfolding ss apply-apply(frule Δ3-implies)
  using finalS-cond-spec by simp

have f4:¬finalS ss4
  using Δ3 unfolding ss apply-apply(frule Δ3-implies)
  using finalS-cond-spec by simp

have vs3 xx = vs4 xx
  using Δ3 lcfgs unfolding ss
  apply-by(frule Δ3-implies, simp)

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react (oor Δ3 Δ1') ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 (oor Δ3 Δ1') ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor Δ3 Δ1') ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 (oor Δ3 Δ1') ss3 ss4 statA ss1 ss2 statO
  proof(rule match12-simpleI, rule disjI1, intro conjI)
    fix ss3' ss4' statA'
    assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfgs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfgs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

  show ¬ isSecO ss3
  using v sa Δ3 unfolding ss by (simp add: Δ3-defs)

```

```

show  $\neg$  isSecO ss4
using v sa  $\Delta 3$  unfolding ss by (simp add:  $\Delta 3$ -defs)

show stat: statA = statA'  $\vee$  statO = Diff
using v sa  $\Delta 3$ 
apply (cases ss3, cases ss4, cases ss1, cases ss2)
apply(cases ss3', cases ss4', clarsimp)
unfolding ss statA' apply clarsimp
apply(simp-all add:  $\Delta 3$ -defs sstatA'-def)
apply(cases statO, simp-all) apply(cases statA, simp-all)
unfolding finalS-defs
by (smt (z3) list.size(3) map-eq-imp-length-eq
      n-not-Suc-n status.exhaust newStat.simps)

have notIO: $\neg$ is-getInput (prog ! pcOf (last cfigs4))  $\neg$ is-Output (prog ! pcOf (last
cfigs4))
using is-Output-pcOf is-getTrustedInput-pcOf
       $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] by auto
have pc:pcOf (last cfigs4) = pcOf (last cfigs3) using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded
ss]] by auto

show oor  $\Delta 3$   $\Delta 1'$   $\infty$  ss3' ss4' statA' ss1 ss2 statO
using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfigs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
    next
      case nonspec-mispred
        then show ?thesis using sa stat  $\Delta 3$  lcfigs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
        next
          case spec-mispred
            then show ?thesis using sa stat  $\Delta 3$  lcfigs unfolding ss apply–
              apply(frule  $\Delta 3$ -implies, clarsimp)
              by (auto simp add:  $\Delta 3$ -defs)
            next
              case spec-normal note sn3 = spec-normal
                show ?thesis
                using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
                  case nonspec-normal
                    then show ?thesis using sa stat  $\Delta 3$  lcfigs sn3 unfolding ss
                      by (simp add:  $\Delta 3$ -defs)
                    next
                      case nonspec-mispred
                        then show ?thesis using sa stat  $\Delta 3$  lcfigs sn3 unfolding ss
                          by (simp add:  $\Delta 3$ -defs)
                      next
                        case spec-mispred

```

```

then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
apply (simp add:  $\Delta 3$ -defs)
by (metis config.sel(1) last-map)
next
case spec-Fence
then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
apply (simp add:  $\Delta 3$ -defs)
by (metis config.sel(1) last-map)
next
case spec-resolve
then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
by (simp add:  $\Delta 3$ -defs)
next
case spec-normal note sn4 = spec-normal
show ?thesis
apply(intro oorI1)
unfolding ss  $\Delta 3$ -def apply- apply(clarify,intro conjI)
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-7[of pc3] apply simp apply(elim disjE)
apply simp-all
by (metis config.collapse config.inject in-set-butlastD last-in-set length-1-butlast
length-map state.sel(2))+
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs)
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-7[of pc3] apply simp apply(elim disjE)
by simp-all
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs )
using cases-7[of pc3] apply simp apply(elim disjE, simp-all)
unfolding array-loc-def by (metis config.sel(2) dist-insert-su last-in-set
state.sel(1) vstore.sel)+
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-7[of pc3] apply simp apply(elim disjE)
apply simp-all by (metis array-loc-def dist-insert-su)+
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-7[of pc3] by(elim disjE, simp-all)
subgoal using sa stat  $\Delta 3$  lcfgs v3 v4 sn3 sn4 unfolding ss hh
apply- apply(frul  $\Delta 3$ -implies) apply(simp-all add:  $\Delta 3$ -defs)
by (metis length-Suc-conv list.size(3)) .
qed(simp-all add: notIO)
next
case spec-Fence note sf3 = spec-Fence
show ?thesis
using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

```

```

    case nonspec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case nonspec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case spec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
    apply (simp add:  $\Delta 3$ -defs)
    by (metis com.disc config.sel(1) last-map)
next
    case spec-resolve
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case spec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sf3 unfolding ss
    apply (simp add:  $\Delta 3$ -defs)
by (metis last-map local.spec-Fence(3) local.spec-normal(1) local.spec-normal(4))

next
    case spec-Fence note sf4 = spec-Fence
    show ?thesis
    apply(intro oorI2)
    unfolding ss  $\Delta 1'$ -defs
    using sa stat  $\Delta 3$  lcfgs v3 v4 sf3 sf4 unfolding ss hh
    apply- by(simp-all add:  $\Delta 3$ -defs  $\Delta 1'$ -defs, blast)
qed(simp-all add: notIO)
next
    case spec-resolve note sr3 = spec-resolve
    show ?thesis
    using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case nonspec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case spec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
next
    case spec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)

```

```

next
  case spec-Fence
  then show ?thesis using sa stat  $\Delta_3$  lcfgs sr3 unfolding ss
  by (simp add:  $\Delta_3$ -defs)
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  apply(intro oorI2)
  using sa stat  $\Delta_3$  lcfgs v3 v4 sr3 sr4 unfolding ss hh
  by(simp add:  $\Delta_3$ -defs  $\Delta_1'$ -defs,metis length-0-conv length-1-butlast
length-map length-pos-if-in-set not-gr-zero)
  qed (simp-all add: notIO)
  qed(simp-all add: notIO[unfolded pc])
qed
qed
qed

```

lemma *step1'*: *unwindIntoCond* Δ_1' Δ_4

proof(*rule* *unwindIntoCond-simpleI*)

fix n ss_3 ss_4 $statA$ ss_1 ss_2 $statO$

assume r : *reachO* ss_3 *reachO* ss_4 *reachV* ss_1 *reachV* ss_2

and Δ_1' : Δ_1' n ss_3 ss_4 $statA$ ss_1 ss_2 $statO$

obtain $pstate_3$ cfg_3 $cfgs_3$ ibT_3 $ibUT_3$ ls_3 **where** ss_3 : $ss_3 = (pstate_3, cfg_3, cfgs_3, ibT_3, ibUT_3, ls_3)$

by (*cases* ss_3 , *auto*)

obtain $pstate_4$ cfg_4 $cfgs_4$ ibT_4 $ibUT_4$ ls_4 **where** ss_4 : $ss_4 = (pstate_4, cfg_4, cfgs_4, ibT_4, ibUT_4, ls_4)$

by (*cases* ss_4 , *auto*)

obtain cfg_1 ibT_1 $ibUT_1$ ls_1 **where** ss_1 : $ss_1 = (cfg_1, ibT_1, ibUT_1, ls_1)$

by (*cases* ss_1 , *auto*)

obtain cfg_2 ibT_2 $ibUT_2$ ls_2 **where** ss_2 : $ss_2 = (cfg_2, ibT_2, ibUT_2, ls_2)$

by (*cases* ss_2 , *auto*)

note $ss = ss_3$ ss_4 ss_1 ss_2

obtain pc_3 vs_3 $avst_3$ h_3 p_3 **where**

cfg_3 : $cfg_3 = Config$ pc_3 (*State* (*Vstore* vs_3) $avst_3$ h_3 p_3)

by (*cases* cfg_3) (*metis* *state.collapse* *vstore.collapse*)

obtain pc_4 vs_4 $avst_4$ h_4 p_4 **where**

cfg_4 : $cfg_4 = Config$ pc_4 (*State* (*Vstore* vs_4) $avst_4$ h_4 p_4)

by (*cases* cfg_4) (*metis* *state.collapse* *vstore.collapse*)

note $cfg = cfg_3$ cfg_4

obtain hh_3 **where** h_3 : $h_3 = Heap$ hh_3 **by**(*cases* h_3 , *auto*)

obtain hh_4 **where** h_4 : $h_4 = Heap$ hh_4 **by**(*cases* h_4 , *auto*)

note $hh = h_3$ h_4

```

have f1:¬finalN ss1
  using Δ1' finalB-pc-iff' unfolding ss cfg finalN-iff-finalB Δ1'-defs
  by simp

have f2:¬finalN ss2
  using Δ1' finalB-pc-iff' unfolding ss cfg finalN-iff-finalB Δ1'-defs
  by simp

have f3:¬finalS ss3
  using Δ1' unfolding ss apply-apply(frul Δ1'-implies)
  using finalS-cond by simp

have f4:¬finalS ss4
  using Δ1' unfolding ss apply-apply(frul Δ1'-implies)
  using finalS-cond by simp

note finals = f1 f2 f3 f4

show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react Δ4 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 Δ4 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 Δ4 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 Δ4 ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

```

```

show eqSec ss1 ss3
  using v sa  $\Delta 1'$  unfolding ss
  by (simp add:  $\Delta 1'$ -defs eqSec-def)

show eqSec ss2 ss4
  using v sa  $\Delta 1'$  unfolding ss
  by (simp add:  $\Delta 1'$ -defs eqSec-def)

show Van.eqAct ss1 ss2
using v sa  $\Delta 1'$  unfolding ss Van.eqAct-def
by (simp-all add:  $\Delta 1'$ -defs)

show match12-12  $\Delta 4$  ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
  using v sa  $\Delta 1'$  sstat unfolding ss cfg statA'
  apply(simp add:  $\Delta 1'$ -defs sstatO'-def sstatA'-def)
  apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
  } note stat = this
  have cfs:cfs3 = [] cfs4 = []
    using is-Output-pcOf is-getTrustedInput-pcOf
       $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss]] by auto

  show  $\Delta 4 \infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO' statO ss1
ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
    case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

    case nonspec-mispred
      then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next

```

```

      case nonspec-normal
      then show ?thesis using sa  $\Delta_1'$  stat v3 v4 nn3 unfolding ss cfg hh
apply clarsimp
  by (auto simp add:  $\Delta_1'$ -defs  $\Delta_4$ -defs)
  qed(simp-all add: cfgs)
  qed(simp-all add: cfgs)
  qed
  qed
  qed
  qed

```

lemma *step: unwindIntoCond* Δ_4 Δ_4

proof(rule *unwindIntoCond-simpleI*)

fix n ss3 ss4 statA ss1 ss2 statO

assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2

and Δ_4 : Δ_4 n ss3 ss4 statA ss1 ss2 statO

obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)

by (cases ss3, auto)

obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)

by (cases ss4, auto)

obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)

by (cases ss1, auto)

obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)

by (cases ss2, auto)

note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 where

cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)

by (cases cfg3) (metis state.collapse vstore.collapse)

obtain pc4 vs4 avst4 h4 p4 where

cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)

by (cases cfg4) (metis state.collapse vstore.collapse)

note cfg = cfg3 cfg4

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)

obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)

note hh = h3 h4

show finalS ss3 = finalS ss4 \wedge finalN ss1 = finalS ss3 \wedge finalN ss2 = finalS ss4

using Δ_4 Opt.final-def Prog.endPC-def finalS-def stepS-endPC endPC-def finalB-endPC

unfolding Δ_4 -defs ss by clarsimp

then show isIntO ss3 = isIntO ss4 by simp

```

show react  $\Delta_4$  ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

  show match1  $\Delta_4$  ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2  $\Delta_4$  ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12  $\Delta_4$  ss3 ss4 statA ss1 ss2 statO
  apply(rule match12-simpleI) using  $\Delta_4$  unfolding ss apply (simp add:  $\Delta_4$ -defs)
  by (simp add: stepS-endPC)
qed
qed

```

lemmas theConds = step0 step1 step2 step3 step1' stepe

proposition rsecure

proof –

```

define m where m: m  $\equiv$  (6::nat)
define  $\Delta_s$  where  $\Delta_s$ :  $\Delta_s \equiv \lambda i::nat.$ 
  if i = 0 then  $\Delta_0$ 
  else if i = 1 then  $\Delta_1$ 
  else if i = 2 then  $\Delta_2$ 
  else if i = 3 then  $\Delta_3$ 
  else if i = 4 then  $\Delta_4$ 
  else  $\Delta_{1'}$ 
define next where next: next  $\equiv \lambda i::nat.$ 
  if i = 0 then {0,1::nat}
  else if i = 1 then {1,2,3,4}
  else if i = 2 then {1}
  else if i = 3 then {3,5}
  else {4}
show ?thesis apply(rule distrib-unwind-rsecure[of m next  $\Delta_s$ ])
  subgoal unfolding m by auto
  subgoal unfolding next m by auto
  subgoal using init unfolding  $\Delta_s$  by auto
  subgoal
    unfolding m next  $\Delta_s$  apply (simp split: if-splits)
    using theConds
    unfolding oor-def oor4-def by auto .
qed
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  $\geq$  0 by auto

```

```

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 :: avname where xx = "i"
definition tt :: 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 aa1  $\neq$  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
unfolding vvars-defs by auto

```

```

fun initAvstore :: avstore  $\Rightarrow$  bool where
  initAvstore (Avstore as) = (as aa1 = (0, size-aa1)  $\wedge$  as aa2 = (size-aa1,
size-aa2))

```

```

fun istate :: state  $\Rightarrow$  bool where
  istate s = (initAvstore (getAvstore s))

```

```

definition prog  $\equiv$ 
[
   $\emptyset$  Start ,
   $\not\! /$  Input U xx ,
   $\not\! /$  tt ::= (N 0) ,
   $\not\! /$  IfJump (Less (V xx) (N NN)) 4 6 ,
   $\not\! /$  vv ::= VA aa1 (N 0) ,
   $\not\! /$  tt ::= Plus (VA aa2 (Times (V vv) (N 512))) (V xx) ,
   $\not\! /$  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
apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
      . . . . .

```

```

lemma cases-thenBranch: (i::pcounter) < 4  $\vee$  i = 4  $\vee$  i = 5  $\vee$  i = 6  $\vee$  i > 6
apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
  subgoal for i apply(cases i, simp-all)
    subgoal for i apply(cases i, simp-all)
      subgoal for i apply(cases i, simp-all)
        subgoal for i apply(cases i, simp-all)
          subgoal for i apply(cases i, simp-all)
            . . . . .

```

```

lemma xx-NN-cases: vs xx < int NN  $\vee$  vs xx  $\geq$  int NN by auto

```

```

lemma is-If-pcOf[simp]:
pcOf cfg < 7  $\implies$  is-IfJump (prog ! (pcOf cfg))  $\longleftrightarrow$  pcOf cfg = 3
apply(cases cfg) subgoal for pc s using cases-6[of pcOf cfg]
by (auto simp: prog-def) .

```

```

lemma is-If-pc[simp]:
pc < 7  $\implies$  is-IfJump (prog ! pc)  $\longleftrightarrow$  pc = 3
using cases-6[of pc]
by (auto simp: prog-def)

```

```

lemma is-If-pcThen[simp]: pcOf cfg  $\in$  {4..6}  $\implies$   $\neg$ is-IfJump (prog ! pcOf cfg)
using cases-thenBranch[of pcOf cfg]
by (auto simp: prog-def)

```

```

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**
 $istate = istate$
by (*standard, simp add: prog-def*)

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

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

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

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

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 $endPC[simp]: endPC = 7$
unfolding $endPC\text{-def}$ **unfolding** $prog\text{-def}$ **by** *auto*

lemma *is-getUntrustedInput-pcOf[simp]: pcOf cfg < 7 \implies is-getInput (prog!(pcOf cfg)) \longleftrightarrow pcOf cfg = 1*
using *cases-6[of pcOf cfg] by (auto simp: prog-def)*

lemma *getUntrustedInput-pcOf[simp]: prog!1 = Input U xx*
by *(auto simp: prog-def)*

lemma *is-getTrustedInput[simp]: is-getInput (prog ! 1)*
unfolding *prog-def by auto*

lemma *getInput-not4[simp]: \neg is-getInput (prog ! 4)*
unfolding *prog-def by auto*

lemma *getInput-not5[simp]: \neg is-getInput (prog ! 5)*
unfolding *prog-def by auto*

lemma *OutputT-not6[simp]: (prog ! 6) = Output U (V tt)*
unfolding *prog-def by auto*

lemma *is-Output-pcOf[simp]: pcOf cfg < 7 \implies is-Output (prog!(pcOf cfg)) \longleftrightarrow pcOf cfg = 6*
using *cases-6[of pcOf cfg] by (auto simp: prog-def)*

lemma *is-Fence-pcOf[simp]: pcOf cfg < 7 \implies prog ! (pcOf cfg) \neq Fence*
using *cases-6[of pcOf cfg] by (auto simp: prog-def)*

lemma *is-Fence-pcThen[simp]: $3 \leq pcOf\ cfg \wedge pcOf\ cfg \leq 5 \implies (prog ! pcOf\ cfg) \neq Fence$*
using *cases-thenBranch[of pcOf cfg]*
by *(auto simp: prog-def)*

lemma *is-Output[simp]: is-Output (prog ! 6)*
unfolding *is-Output-def prog-def by auto*

lemma *getInput-not[intro]: is-getInput (prog ! 4) \implies False* **unfolding** *prog-def by simp*

lemma *Output-not4[intro]: is-Output (prog ! 4) \implies False* **unfolding** *prog-def by simp*

lemma *Fence-not4[intro]: prog ! 4 = Fence \implies False* **unfolding** *prog-def by simp*

lemma *getInput-not55[intro]: is-getInput (prog ! 5) \implies False* **unfolding** *prog-def by simp*

lemma *Output-not5[intro]: is-Output (prog ! 5) \implies False* **unfolding** *prog-def by simp*

lemma *Fence-not5[intro]: prog ! 5 = Fence \implies False* **unfolding** *prog-def by simp*

lemma *Jump-not6: \neg is-IfJump (prog ! 6)* **unfolding** *prog-def by simp*

lemma *isSecV-pcOf[simp]*:
 $isSecV (cfg, ibT, ibUT) \longleftrightarrow pcOf\ cfg = 0$
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf[simp]*:
 $isSecO (pstate, cfg, cfs, ibT, ibUT, ls) \longleftrightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
using *isSecO-def* **by** *simp*

lemma *inputT-not[simp]*: $pcOf\ cfg < 7 \implies$
 $(prog\ !\ pcOf\ cfg) \neq Input\ T\ inp$
apply(*cases\ cfg*) **subgoal for** *pc\ s* **using** *cases-6[of\ pcOf\ cfg]*
by (*auto\ simp: prog-def*) .

lemma *getActV-pcOf[simp]*:
 $pcOf\ cfg < 7 \implies$
 $getActV (cfg, ibT, ibUT, ls) =$
(if pcOf\ cfg = 1 then lhd\ ibUT else \perp)
apply(*subst\ getActV-simps*) **unfolding** *prog-def*
apply *simp*
using *getActV-simps not-is-getTrustedInput-getActV prog-def* **by** *auto*

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
)
apply(*subst\ getObsV-simps*)
unfolding *prog-def* **apply** *simp*
using *getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def*
by (*auto, simp*)

lemma *getObsV-pcOf6[simp]*:
 $pcOf\ cfg = 6 \implies$
 $getObsV (cfg, ibT, ibUT, ls) =$
 $(outOf (prog!(pcOf\ cfg)) (stateOf\ cfg), ls)$

by *simp*

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)
apply(*subst\ getActO-simps*)

apply(cases cfigs, auto)
unfolding prog-def **apply** simp
using getActV-simps getActV-pcOf prog-def **by** presburger

lemma getObsO-pcOf[simp]:
 $pcOf\ cfg < 7 \implies$
 $getObsO\ (pstate, cfig, cfigs, ibT, ibUT, ls) =$
 (if (pcOf cfig = 6 \wedge cfigs = []) then
 (outOf (prog!(pcOf cfig)) (stateOf cfig), ls)
 else \perp
)
apply(subst getObsO-simps)
apply(cases cfigs, auto)
unfolding prog-def
using getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def **by** presburger

lemma eqSec-pcOf[simp]:
 $eqSec\ (cfg1, ibT, ibUT1, ls1)\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfigs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
unfolding eqSec-def **by** simp

lemma nextB-pc0[simp]:
 $nextB\ (Config\ 0\ s, ibT, ibUT) =$
 $(Config\ 1\ s, ibT, ibUT)$
apply(subst nextB-Start-Skip-Fence)
unfolding endPC-def **unfolding** prog-def **by** auto

lemma readLocs-pc0[simp]:
 $readLocs\ (Config\ 0\ s) = \{\}$
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

lemma nextB-pc1[simp]:
 $ibUT \neq LNil \implies nextB\ (Config\ 1\ (State\ (Vstore\ vs)\ avst\ h\ p), ibT, ibUT) =$
 $(Config\ 2\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ h\ p), ibT, ltl\ ibUT)$
apply(subst nextB-getUntrustedInput')
unfolding endPC-def **unfolding** prog-def **by** auto

lemma readLocs-pc1[simp]:
 $readLocs\ (Config\ 1\ s) = \{\}$
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

lemma *nextB-pc1* [simp]:
 $ibUT \neq LNil \implies nextB (Config (Suc 0) (State (Vstore vs) avst h p), ibT, ibUT)$
 $=$
 $(Config\ 2 (State (Vstore (vs(xx := lhd\ ibUT)))) avst\ h\ p), ibT, ltl\ ibUT)$
apply(subst *nextB-getUntrustedInput*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1* [simp]:
 $readLocs (Config (Suc 0) s) = \{\}$
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc2* [simp]:
 $nextB (Config\ 2 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT) =$
 $(Config\ 3 (State (Vstore (vs(tt := 0))) avst\ h\ p), ibT, ibUT)$
apply(subst *nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc2* [simp]:
 $readLocs (Config\ 2\ s) = \{\}$
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3-then* [simp]:
 $vs\ xx < int\ NN \implies$
 $nextB (Config\ 3 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT) =$
 $(Config\ 4 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT)$
apply(subst *nextB-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3-else* [simp]:
 $vs\ xx \geq int\ NN \implies$
 $nextB (Config\ 3 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT) =$
 $(Config\ 6 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT)$
apply(subst *nextB-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc3*:
 $nextB (Config\ 3 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT) =$
 $(Config (if\ vs\ xx < int\ NN\ then\ 4\ else\ 6) (State (Vstore\ vs) avst\ h\ p), ibT, ibUT)$
by(cases $vs\ xx < int\ NN$, *auto*)

lemma *nextM-pc3-then* [simp]:
 $vs\ xx \geq int\ NN \implies$
 $nextM (Config\ 3 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT) =$
 $(Config\ 4 (State (Vstore\ vs) avst\ h\ p), ibT, ibUT)$
apply(subst *nextM-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextM-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
by(*cases vs xx < int NN*, *auto*)

lemma *readLocs-pc3*[simp]:
readLocs (*Config 3 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc4*[simp]:
readLocs (*Config 4* (*State* (*Vstore vs*) *avst h p*)) = {*array-loc aa1 0 avst*}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc5*[simp]:
readLocs (*Config 5* (*State* (*Vstore vs*) *avst h p*)) = {*array-loc aa2 (nat (vs vv * 512)) avst*}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc6*[simp]:
nextB (*Config 6 s*, *ibT*, *ibUT*) = (*Config 7 s*, *ibT*, *ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc6*[*simp*]:
readLocs (Config 6 (State (Vstore vs) avst h p)) = {}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-stepB-pc*:
 $pc < 7 \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-6*[*of pc*] **apply** *safe*
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def, metis llist.collapse*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) **subgoal apply** *simp* **apply**(*subst stepB.simps*)
unfolding *endPC-def*
by (*simp add: prog-def*)

subgoal apply(*cases vs xx < NN*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .
subgoal apply(*cases vs xx < NN*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal by auto
subgoal by auto

...

lemma nextB-avst-consistent-aux:

$4 \leq pc \wedge pc \leq 6 \implies$
 $(nextB (Config pc (State (Vstore vs) avst (Heap h) p), ibT, ibUT)) = (Config pc'$
 $(State (Vstore vs') avst' (Heap h') p'), ibT, ibUT')) \implies$
 $avst = avst' \wedge$
 $vs\ xx = vs'\ xx \wedge$
 $h = h'$
using cases-thenBranch[of pc]
apply safe
apply simp-all by auto

lemma nextB-avst-consistent:

$4 \leq pcOf\ cfg \wedge pcOf\ cfg \leq 6 \implies$
 $(nextB (cfg, ibT, ibUT)) = (cfg', ibT, ibUT') \implies$
 $(getAvstore (stateOf\ cfg)) = (getAvstore (stateOf\ cfg')) \wedge$
 $(getHheap (stateOf\ cfg)) = (getHheap (stateOf\ cfg')) \wedge$
 $vstore (getVstore (stateOf\ cfg))\ xx = vstore (getVstore (stateOf\ cfg'))\ xx$
apply(cases cfg) **subgoal for pc s**
apply(cases s) **subgoal for vstore avst heap-h p**
apply (cases heap-h, cases vstore, cases avst) **subgoal for h vs**
apply(cases cfg') **subgoal for pc' s'**
apply(cases s') **subgoal for vstore' avst' heap-h' p'**
apply (cases heap-h', cases vstore', cases avst') **subgoal for h vs**
using nextB-avst-consistent-aux apply simp
by blast

lemma nextB-pcs-consistent:

$4 \leq pcOf\ cfg1 \wedge pcOf\ cfg1 \leq 6 \implies pcOf\ cfg1 = pcOf\ cfg2 \implies$
 $(nextB (cfg1, ibT1, ibUT1)) = (cfg1', ibT1', ibUT1') \implies$
 $(nextB (cfg2, ibT2, ibUT2)) = (cfg2', ibT2', ibUT2') \implies$
 $pcOf\ cfg1' = pcOf\ cfg2'$
apply (cases cfg1, cases cfg2, cases cfg1', cases cfg2')
subgoal for pc1 s1 pc2 s2 pc1' s1' pc2' s2'
apply(cases s1, cases s2, cases s1', cases s2')
subgoal for vs1 avst1 h1 p1 vs2 avst2 h2 p2
 $vs1'\ avst1'\ h1'\ p1'\ vs2'\ avst2'\ h2'\ p2'$
apply(cases vs1, cases vs2, cases h1, cases h2)
using cases-6[of pcOf cfg1] **apply safe**
by simp-all . .

lemma *not-finalB*:

$pc < \gamma \implies (pc = 1 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
using *nextB-stepB-pc* **by** (*simp add: stepB-iff-nextB*)

lemma *finalB-pc-iff'*:

$pc < \gamma \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil)$
subgoal **apply** *safe*
subgoal **using** *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)
subgoal **using** *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)
subgoal **using** *finalB-iff getUntrustedInput-pcOf* **by** *auto . .*

lemma *finalB-pc-iff*:

$pc \leq \gamma \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 1 \wedge ibUT = LNil \vee pc = \gamma)$
using *cases-6[of pc]* **apply** (*elim disjE, simp add: finalB-def*)
subgoal **by** (*meson final-def stebB-0*)
by (*simp add: finalB-pc-iff' finalB-endPC*)+

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

$pcOf\ cfg \leq \gamma \implies$
 $finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibUT = LNil \vee pcOf\ cfg = \gamma)$
by (*metis config.exhaust config.sel(1) finalB-pc-iff*)

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

apply(*cases cfg*)
subgoal **for** *pc s* **apply**(*cases s*)
subgoal **for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal **for** *vs as h*
using *cases-6[of pc]* **apply**(*elim disjE*) **unfolding** *finalS-defs*
subgoal **using** *nonspec-normal[of [] Config pc (State (Vstore vs) avst hh p)*
pstate pstate ibT ibUT
Config 1 (State (Vstore vs) avst hh p)
ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore
vs) avst hh p)) ls]
using *is-If-pc* **by** *force*

subgoal **apply**(*frule nonspec-normal[of cfigs Config pc (State (Vstore vs) avst*
hh p)

pstate pstate ibT ibUT
Config 2 (State (Vstore (vs(xx:= lhd ibUT))) avst hh
p)

$ibT \text{ ltl } ibUT \ [] \text{ } ls \cup \text{ readLocs } (Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)) \text{ } ls])$

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of cfigs $Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$])

$pstate \text{ } pstate \text{ } ibT \text{ } ibUT$

$Config \text{ } 3 \text{ } (State \text{ } (Vstore \text{ } (vs(tt:= 0))) \text{ } avst \text{ } hh \text{ } p)$

$ibT \text{ } ibUT \ [] \text{ } ls \cup \text{ readLocs } (Config \text{ } pc \text{ } (State \text{ } (Vstore$

$vs) \text{ } avst \text{ } hh \text{ } p)) \text{ } ls])$

prefer 7 subgoal by metis by simp-all

subgoal apply(*cases mispred pstate [3]*)

subgoal apply(*frule nonspec-mispred*[of cfigs $Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$])

$pstate \text{ } update \text{ } pstate \text{ } [pcOf \text{ } (Config \text{ } pc \text{ } (State$

$(Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p))]$

$ibT \text{ } ibUT \text{ } Config \text{ } (if \text{ } vs \text{ } xx < NN \text{ } then \text{ } 4 \text{ } else \text{ } 6)$

$(State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$

$ibT \text{ } ibUT \text{ } Config \text{ } (if \text{ } vs \text{ } xx < NN \text{ } then \text{ } 6 \text{ } else \text{ } 4)$

$(State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$

$ibT \text{ } ibUT \text{ } [Config \text{ } (if \text{ } vs \text{ } xx < NN \text{ } then \text{ } 6 \text{ } else$

$4) \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)]$

$ls \cup \text{ readLocs } (Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs)$

$avst \text{ } hh \text{ } p)) \text{ } ls])$

prefer 9 subgoal by metis by (simp add: finalM-iff)+

subgoal apply(*frule nonspec-normal*[of cfigs $Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$])

$pstate \text{ } pstate \text{ } ibT \text{ } ibUT$

$Config \text{ } (if \text{ } vs \text{ } xx < NN \text{ } then \text{ } 4 \text{ } else \text{ } 6) \text{ } (State \text{ } (Vstore$

$vs) \text{ } avst \text{ } hh \text{ } p)$

$ibT \text{ } ibUT \ [] \text{ } ls \cup \text{ readLocs } (Config \text{ } pc \text{ } (State \text{ } (Vstore$

$vs) \text{ } avst \text{ } hh \text{ } p)) \text{ } ls])$

prefer 7 subgoal by metis by simp-all .

subgoal apply(*frule nonspec-normal*[of cfigs $Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$])

$pstate \text{ } pstate \text{ } ibT \text{ } ibUT$

$(let \text{ } l = (array-loc \text{ } aa1 \text{ } 0 \text{ } avst)$

$in \text{ } (Config \text{ } 5 \text{ } (State \text{ } (Vstore \text{ } (vs(vv := h \text{ } l))) \text{ } avst \text{ } hh \text{ } p)))$

$ibT \text{ } ibUT \ [] \text{ } ls \cup \text{ readLocs } (Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst$

$hh \text{ } p)) \text{ } ls])$

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of cfigs $Config \text{ } pc \text{ } (State \text{ } (Vstore \text{ } vs) \text{ } avst \text{ } hh \text{ } p)$])

$pstate \text{ } pstate \text{ } ibT \text{ } ibUT$

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

$hh\ p))$
 $in\ (Config\ 6\ (State\ (Vstore\ (vs(tt := h\ l + vs\ xx)))\ avst$
 $hh\ p))\ ls])$
 $ibT\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore\ vs)\ avst$
 $hh\ p))\ ls])$
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

$pstate\ pstate\ ibT\ ibUT$
 $Config\ 7\ (State\ (Vstore\ vs)\ avst\ hh\ p)$
 $ibT\ ibUT\ []\ ls\ ls])$
prefer 7 subgoal by metis by simp-all
by simp-all . . .

lemma *finalS-cond-spec*:

$pcOf\ cfg < 7 \implies$
 $((pcOf\ (last\ cfgs) = 4 \vee pcOf\ (last\ cfgs) = 5 \vee pcOf\ (last\ cfgs) = 6) \wedge pcOf$
 $cfg = 6) \vee (pcOf\ (last\ cfgs) = 6 \wedge pcOf\ cfg = 4) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg\ finalS\ (pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls)$

apply(*cases cfg*)

subgoal for *pc s* **apply**(*cases s*)

subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)

subgoal for *vs as h* **apply**(*cases last cfgs*)

subgoal for *pcs ss* **apply**(*cases ss*)

subgoal for *vsts avsts hhs ps* **apply**(*cases vsts, cases avsts, cases hhs, simp*)

subgoal for *vss ass hs* **apply**(*elim disjE, elim conjE, elim disjE, simp*)

unfolding *finalS-defs*

subgoal apply(*cases \neg resolve pstate [6, 4]*)

subgoal apply(*rule notI,*

erule allE[of - (*pstate, Config 6 (State (Vstore vs) (Avstore as) (Heap h) p)*),

[*Config 5 (State (Vstore (vss(vv := hs (array-loc aa1 (nat 0) avsts)))) avsts hhs ps*]],

ibT, ibUT, ls \cup readLocs (last cfgs)], *erule notE,*

rule spec-normal[of - - - - - *Config 5 (State (Vstore (vss(vv := hs (array-loc aa1 (nat 0) avsts)))) avsts hhs ps*]])

by auto

apply(*rule notI,*

erule allE[of - (*update pstate [6, 4], Config 6 (State (Vstore vs) (Avstore as) (Heap h) p)*),

$[],$
ibT, ibUT, ls]])

by(*erule notE,*

rule spec-resolve, auto)

subgoal apply(*cases \neg resolve pstate [6, 5]*)

subgoal apply(*rule notI,*

```

    erule allE[of - (pstate, Config 6 (State (Vstore vs) (Avstore as) (Heap h)
p),
      [Config 6 (State (Vstore (vss(tt := hs (array-loc aa2 (nat
(vss vv * 512)) avsts) + vss xx))) avsts hhs ps)],
      ibT, ibUT, ls  $\cup$  readLocs (last cfgs)], erule notE,
      rule spec-normal[of - - - - Config 6 (State (Vstore (vss(tt
:= hs (array-loc aa2 (nat (vss vv * 512)) avsts) + vss xx))) avsts hhs ps)])

    prefer 7 apply auto[1]
    by auto

    subgoal apply(rule notI,
      erule allE[of - (update pstate (6 # map pcOf cfgs), Config 6 (State (Vstore vs)
(Avstore as) (Heap h) p),
        [], ibT, ibUT, ls)])
    by(erule notE, rule spec-resolve, auto) .

    subgoal apply(rule notI,
      erule allE[of - (update pstate (6 # map pcOf cfgs), Config 6 (State (Vstore vs)
(Avstore as) (Heap h) p),
        [], ibT, ibUT, ls)])
    by(erule notE, rule spec-resolve, auto)
    subgoal apply(rule notI,
      erule allE[of - (update pstate (4 # map pcOf cfgs), Config 4 (State (Vstore vs)
(Avstore as) (Heap h) p),
        [], ibT, ibUT, ls)])
    by(erule notE, rule spec-resolve, auto) . . . . .

end

```

11.2 Proof

```

theory Fun4-secure
  imports Fun4
begin

```

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

definition $same\text{-}xx\text{-}cp\ cfg1\ cfg2 \equiv$
 $vstore\ (getVstore\ (stateOf\ cfg1))\ xx = vstore\ (getVstore\ (stateOf\ cfg2))\ xx$
 $\wedge\ vstore\ (getVstore\ (stateOf\ cfg1))\ xx = 0$

definition $common\text{-}memory\ cfg\ cfg'\ cfgs' \equiv$
 $array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg)) = array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg')) \wedge$
 $(\forall\ cfg'' \in set\ cfgs'.\ array\text{-}base\ aa1\ (getAvstore\ (stateOf\ cfg'')) = array\text{-}base\ aa1$

$$\begin{aligned}
& (\text{getAvstore } (\text{stateOf } \text{cfg})) \wedge \\
& \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg})) = \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \\
& \text{cfg}')) \wedge \\
& (\forall \text{cfg}'' \in \text{set } \text{cfgs}'. \text{array-base aa2 } (\text{getAvstore } (\text{stateOf } \text{cfg}'')) = \text{array-base aa2} \\
& (\text{getAvstore } (\text{stateOf } \text{cfg}))) \wedge \\
& (\text{getHheap } (\text{stateOf } \text{cfg})) = (\text{getHheap } (\text{stateOf } \text{cfg}')) \wedge \\
& (\forall \text{cfg}'' \in \text{set } \text{cfgs}'. \text{getHheap } (\text{stateOf } \text{cfg}) = (\text{getHheap } (\text{stateOf } \text{cfg}''))) \wedge \\
& (\text{getAvstore } (\text{stateOf } \text{cfg})) = (\text{getAvstore } (\text{stateOf } \text{cfg}'))
\end{aligned}$$

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

pcOf *cfg3* = *pcOf* *cfg4* \wedge *map* *pcOf* *cfgs3* = *map* *pcOf* *cfgs4* \wedge

pcOf *cfg3* \in *PC* \wedge *pcOf* '(*set* *cfgs3*) \subseteq *PC* \wedge

~~///A///B///C///D///E///F///G///H///I///J///K///L///M///N///O///P///Q///R///S///T///U///V///W///X///Y///Z///~~
common-memory *cfg1* *cfg3* *cfgs3* \wedge

~~///a///b///c///d///e///f///g///h///i///j///k///l///m///n///o///p///q///r///s///t///u///v///w///x///y///z///~~
common-memory *cfg2* *cfg4* *cfgs4* \wedge

($\forall n \geq 0$. *array-loc* *aa1* 0 (*getAvstore* (*stateOf* *cfg2*)) \neq *array-loc* *aa2* *n* (*getAvstore* (*stateOf* *cfg2*))) \wedge

array-loc *aa1* 0 (*getAvstore* (*stateOf* *cfg1*)) \neq *array-loc* *aa2* *n* (*getAvstore* (*stateOf* *cfg1*))) \wedge

```

///
array-base aa1 (getAvstore (stateOf cfg3)) = array-base aa1 (getAvstore (stateOf
cfg4)) ∧
(∀ cfg3' ∈ set cfigs3. array-base aa1 (getAvstore (stateOf cfg3')) = array-base aa1
(getAvstore (stateOf cfg3))) ∧
(∀ cfg4' ∈ set cfigs4. 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 cfigs3. array-base aa2 (getAvstore (stateOf cfg3')) = array-base aa2
(getAvstore (stateOf cfg3))) ∧
(∀ cfg4' ∈ set cfigs4. array-base aa2 (getAvstore (stateOf cfg4')) = array-base aa2
(getAvstore (stateOf cfg4))) ∧
///
(statA = Diff → statO = Diff))

```

lemmas *common-strat1-defs = common-strat1-def common-memory-def*

definition *common :: enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status ⇒ bool*

where

```

common = (λ(num::enat)
  (pstate3, cfg3, cfigs3, ibT3, ibUT3, ls3)
  (pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)
  statA
  (cfg1, ibT1, ibUT1, ls1)
  (cfg2, ibT2, ibUT2, ls2)
  statO.
  (pstate3 = pstate4 ∧

  (num = (endPC - pcOf cfg1) ∨ num = ∞) ∧

```

```

  (pcOf cfg1 = pcOf cfg2 ∧

```

```

  (pcOf cfg3 = pcOf cfg4 ∧
  map pcOf cfigs3 = map pcOf cfigs4 ∧
  pcOf cfg3 ∈ PC ∧ pcOf (set cfigs3) ⊆ PC ∧
  pcOf cfg1 ∈ PC ∧

```

```

  common-memory cfg1 cfg3 cfigs3 ∧

```

```

  common-memory cfg2 cfg4 cfigs4 ∧

```

```

  (∀ 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))) ∧
All traces have same base addresses. Avoid. Maybe this is also worth being extracted
as a predicate.
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)
))

```

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)))

```

unfolding *common-defs PC-def*

by *force*

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**

```

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 num (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
 statA
 (cfg1, ibT1, ibUT1, ls1)
 (cfg2, ibT2, ibUT2, ls2)
 statO \wedge

~~head of the buffers are equal counterpartwise~~
 (llength ibUT1 = ∞ \wedge llength ibUT2 = ∞ \wedge
 llength ibUT3 = ∞ \wedge llength ibUT4 = ∞) \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

~~size of the buffers is equal, cfg1 = cfg3 and cfg2 = cfg4 in the ordering state~~
 cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 ls1 = {} \wedge ls2 = {} \wedge
 noMisSpec cfs3

))

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

~~head of the buffers are equal counterpartwise~~
 (llength ibUT1 = ∞ \wedge llength ibUT2 = ∞ \wedge
 llength ibUT3 = ∞ \wedge llength ibUT4 = ∞) \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

~~size of the buffers is equal, cfg1 = cfg3 and cfg2 = cfg4 in the ordering state~~
 cfg1 = cfg3 \wedge cfg2 = cfg4 \wedge
 ls1 = ls3 \wedge ls2 = ls4 \wedge
 ls1 = {} \wedge ls2 = {} \wedge

```

noMisSpec cfgs3
)
unfolding  $\Delta 0$ -defs' apply(clarsimp, standard)
subgoal by (smt (verit) infinity-ne-i0 llength-LNil)
subgoal by (smt (verit)) .

lemmas  $\Delta 0$ -defs =  $\Delta 0$ -def2 common-defs PC-def beforeInput-def noMisSpec-def

lemma  $\Delta 0$ -implies:  $\Delta 0$  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 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
unfolding  $\Delta 0$ -defs
apply(intro conjI)
apply (simp-all)
by (metis Nil-is-map-conv)

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-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  $\in$  afterInput  $\wedge$ 
  same-var-o xx cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
  vstore (getVstore (stateOf cfg3)) xx < NN  $\wedge$ 

  ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 
  noMisSpec cfgs3
))

```

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  $\implies$ 
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
unfolding  $\Delta 1$ -defs
apply(intro conjI) apply simp-all
by (metis map-is-Nil-conv)
```

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, 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 cfg1 = pcOf cfg3  $\wedge$ 

pcOf (last cfs3) = elseBranch  $\wedge$ 
same-var-o xx cfg3 cfs3 cfg4 cfs4  $\wedge$ 
vstore (getVstore (stateOf cfg3)) xx < NN  $\wedge$ 
ls1 = ls3  $\wedge$  ls2 = ls4  $\wedge$ 
misSpecL1 cfs3
))
```

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, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO  $\implies$ 
pcOf (last cfs3) = 6  $\wedge$  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
apply(intro conjI)
unfolding  $\Delta 2$ -defs apply simp-all
apply (metis last-map map-is-Nil-conv)
by (metis length-map)

```

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 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 cfg3 cfs3 cfg4 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 cfgs3
))
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, cfgs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfgs4, 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  $\wedge$ 
cfgs3 = []  $\wedge$  pcOf cfg3 < 7  $\wedge$ 
cfgs4 = []  $\wedge$  pcOf cfg4 < 7
unfolding  $\Delta 1'$ -defs
apply (intro conjI)
apply simp-all
using Suc-lessI startOfThenBranch-def verit-eq-simplify(10) zero-neq-numeral
apply linarith
by (metis list.map-disc-iff)

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 num (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  $\wedge$  cfgs3  $\neq$  []  $\wedge$ 
pcOf (last cfgs3)  $\in$  inThenBranch  $\wedge$ 
pcOf (last cfgs4) = pcOf (last cfgs3)  $\wedge$ 
pcOf cfg1 = pcOf (last cfgs3)  $\wedge$ 

```

$same\text{-}var\text{-}o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $(getAvstore\ (stateOf\ cfg3)) = (getAvstore\ (stateOf\ (last\ cfs3))) \wedge$
 $(getAvstore\ (stateOf\ cfg4)) = (getAvstore\ (stateOf\ (last\ cfs4))) \wedge$

$same\text{-}xx\text{-}cp\ cfg1\ cfg2 \wedge$
 $ls1 = ls3 \wedge ls2 = ls4 \wedge$

$vstore\ (getVstore\ (stateOf\ cfg3))\ tt = vstore\ (getVstore\ (stateOf\ cfg4))\ tt \wedge$

$vstore\ (getVstore\ (stateOf\ cfg3))\ xx \geq NN \wedge$

$(pcOf\ cfg1 = 4 \longrightarrow ls1 = \{\} \wedge ls2 = \{\}) \wedge$
 $(pcOf\ cfg1 \leq 5 \longrightarrow ls1 \subseteq \{array\text{-}loc\ aa1\ 0\ (getAvstore\ (stateOf\ cfg1))\}$
 $\wedge ls2 \subseteq \{array\text{-}loc\ aa1\ 0\ (getAvstore\ (stateOf\ cfg2))\}$
 $\wedge ls3 = ls4) \wedge$

$(pcOf\ cfg1 > 4 \longrightarrow same\text{-}var\ vv\ cfg1\ (last\ cfs3) \wedge same\text{-}var\ vv\ cfg2\ (last\ cfs4))$
 \wedge
 $misSpecL1\ cfs3$
 $)$

lemmas $\Delta 3'\text{-}defs = \Delta 3'\text{-}def\ common\text{-}defs\ PC\text{-}def\ elseBranch\text{-}def$
 $inThenBranch\text{-}def\ startOfThenBranch\text{-}def$
 $same\text{-}var\text{-}o\text{-}def\ same\text{-}xx\text{-}cp\text{-}def\ misSpecL1\text{-}def\ same\text{-}var\text{-}def$

lemma $\Delta 3'\text{-}implies: \Delta 3'\ num\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg1 < 7 \wedge pcOf\ cfg1 \neq Suc\ 0 \wedge$
 $pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $pcOf\ cfg3 < 7 \wedge pcOf\ cfg4 < 7 \wedge$
 $pcOf\ (last\ cfs3) = pcOf\ (last\ cfs4) \wedge$
 $(pcOf\ (last\ cfs3) = 4 \vee pcOf\ (last\ cfs3) = 5 \vee pcOf\ (last\ cfs3) = 6) \wedge pcOf$
 $cfg3 = 6 \wedge$
 $pcOf\ cfg3 = pcOf\ cfg4 \wedge$
 $length\ cfs3 = Suc\ 0 \wedge$
 $length\ cfs3 = length\ cfs4$

unfolding $\Delta 3'\text{-}defs$
apply $(intro\ conjI)$
apply $simp\text{-}all$
by $(metis\ cases\text{-}thenBranch\ le\text{-}neq\text{-}implies\text{-}less\ less\text{-}SucI\ not\text{-}less\text{-}eq\ length\text{-}map)+$

definition $\Delta e :: enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV \Rightarrow stateV \Rightarrow status$

```

⇒ bool where
  Δe = (λ(num::enat) (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)
    (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)
    statA
    (cfg1, ibT1, ibUT1, ls1)
    (cfg2, ibT2, ibUT2, ls2)
    statO.
    ((num = (endPC - pcOf cfg1) ∨ num = ∞) ∧
     pcOf cfg3 = endPC ∧ pcOf cfg4 = endPC ∧ cfgs3 = [] ∧ cfgs4 = [] ∧
     pcOf cfg1 = endPC ∧ pcOf cfg2 = endPC))

```

lemmas Δe-defs = Δe-def common-def endPC

context array-nempty

begin

lemma *init*: *initCond* Δ0

unfolding *initCond-def* apply (*intro allI*)

subgoal for *s3 s4* apply (*cases s3, cases s4*)

subgoal for *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4*

apply *safe*

apply *clarsimp*

apply (*cases lhd ibUT3 < NN*)

subgoal

apply (*cases getAvstore (stateOf cfg3), cases getAvstore (stateOf cfg4)*)

unfolding Δ0-defs

unfolding *array-base-def array-loc-def*

using *aa1* by *auto*

subgoal

apply (*cases getAvstore (stateOf cfg3), cases getAvstore (stateOf cfg4)*)

unfolding Δ0-defs'

unfolding *array-base-def array-loc-def*

using *aa1* apply (*simp split: avstore.splits*)

apply (*rule exI[of - cfg3]*) using *ex-llength-infty* by *auto*

...

lemma *step0*: *unwindIntoCond* Δ0 (*oor3* Δ0 Δ1 Δ1')

proof (*rule unwindIntoCond-simpleI*)

fix *n ss3 ss4 statA ss1 ss2 statO*

assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*

and Δ0: Δ0 *n ss3 ss4 statA ss1 ss2 statO*

obtain *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3* where *ss3*: *ss3* = (*pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3*)

by (*cases ss3, auto*)

obtain *pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4* where *ss4*: *ss4* = (*pstate4, cfg4, cfgs4,*

```

ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where
    cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
    by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
    cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
    by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  obtain hh3 where h3: h3 = Heap hh3 by (cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by (cases h4, auto)
  note hh = h3 h4

  have f1: ¬finalN ss1
    using Δ0 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ0-defs
    by simp

  have f2: ¬finalN ss2
    using Δ0 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ0-defs
    by simp

  have f3: ¬finalS ss3
    using Δ0 unfolding ss apply-apply (frule Δ0-implies)
    using finalS-cond by simp

  have f4: ¬finalS ss4
    using Δ0 unfolding ss apply-apply (frule Δ0-implies)
    using finalS-cond by simp

  note finals = f1 f2 f3 f4
  show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
    using finals by auto

  then show isIntO ss3 = isIntO ss4 by simp

  show react (oor3 Δ0 Δ1 Δ1') ss3 ss4 statA ss1 ss2 statO
    unfolding react-def proof (intro conjI)

```

```

show match1 (oor3  $\Delta 0$   $\Delta 1$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor3  $\Delta 0$   $\Delta 1$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor3  $\Delta 0$   $\Delta 1$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
    cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
    cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain pc3 vs3 avst3 h3 p3 where
    cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
    cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  show eqSec ss1 ss3
    using v  $\Delta 0$  unfolding ss by (simp add:  $\Delta 0$ -defs)

  show eqSec ss2 ss4
    using v  $\Delta 0$  unfolding ss
    apply (simp add:  $\Delta 0$ -defs) by (metis length-0-conv length-map)

  show saO: Van.eqAct ss1 ss2
  using v sa  $\Delta 0$  unfolding ss
  unfolding Opt.eqAct-def Van.eqAct-def
  apply(simp-all add:  $\Delta 0$ -defs)
  by (metis enat.distinct(2) f3 list.map-disc-iff llength-LNil ss3 zero-enat-def)

  show match12-12 (oor3  $\Delta 0$   $\Delta 1$   $\Delta 1'$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
    conjI impI)

    show validTransV (ss1, nextN ss1)
      by (simp add: f1 nextN-stepN)

```

```

show validTransV (ss2, nextN ss2)
  by (simp add: f2 nextN-stepN)

{assume sstat: statA' = Diff
 show sstatO' statO ss1 ss2 = Diff
 using v sa  $\Delta 0$  sstat unfolding ss cfg statA' apply simp
 apply(simp add:  $\Delta 0$ -defs sstatO'-def sstatA'-def finalS-def final-def)
 using cases-6[of pc3] apply(elim disjE)
 apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
 apply(cases statO, simp-all) apply (cases statA, simp-all)
 apply (smt (z3) status.distinct newStat.simps)
 using newStat.simps by (smt (z3) status.exhaust)
} note stat = this

have cfigs:cfigs3 = [] cfigs4 = [] using  $\Delta 0$  unfolding ss apply-by(frule
 $\Delta 0$ -implies, auto)+

show oor3  $\Delta 0$   $\Delta 1$   $\Delta 1' \infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa  $\Delta 0$  stat unfolding ss apply-by apply(frule
 $\Delta 0$ -implies)
    by (simp add:  $\Delta 0$ -defs)
  next
  case spec-normal
  then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
  next
  case spec-mispred
  then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
  next
  case spec-Fence
  then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
  next
  case spec-resolve
  then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
  next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis
    using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-mispred
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
    next
    case spec-normal
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:

```

```

Δ0-defs)
  next
  case spec-mispred
  then show ?thesis using sa Δ0 stat nn3 unfolding ss by (simp add:
Δ0-defs)
  next
  case spec-Fence
  then show ?thesis using sa Δ0 stat nn3 unfolding ss by (simp add:
Δ0-defs)
  next
  case spec-resolve
  then show ?thesis using sa Δ0 stat nn3 unfolding ss by (simp add:
Δ0-defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  show ?thesis using sa saO Δ0 stat v3 v4 nn3 nn4 f4
  unfolding ss cfg Opt.eqAct-def apply clarsimp
  apply(cases pc3 = 0)
  subgoal apply(rule oor3I1)
  apply (simp add: Δ0-defs) by (metis config.sel(2) state.sel(2))
  subgoal apply(subgoal-tac pc4 = 1)
  defer subgoal by (simp add: Δ0-defs)
  subgoal using xx-NN-cases[of vstore (getVstore (stateOf cfg3'))]
apply(elim disjE)
  subgoal apply(rule oor3I2)
  by (simp add: Δ0-defs Δ1-defs, metis)
  subgoal apply(rule oor3I3)
  apply (simp add: Δ0-defs Δ1'-defs)
  apply(intro conjI, metis+)
  apply blast by fastforce+
  ...
  qed(simp-all add: cfgs)
  qed(simp-all add: cfgs)
  qed
  qed
  qed
  qed

```

lemma *step1: unwindIntoCond Δ1 (oor3 Δ1 Δ2 Δe)*

proof(rule *unwindIntoCond-simpleI*)

fix *n ss3 ss4 statA ss1 ss2 statO*

assume *r: reachO ss3 reachO ss4 reachV ss1 reachV ss2*

and *Δ1: Δ1 n ss3 ss4 statA ss1 ss2 statO*

obtain *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3* **where** *ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)*

by (*cases ss3, auto*)

obtain *pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4* **where** *ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)*

by (*cases ss4, auto*)
obtain *cfg1 ibT1 ibUT1 ls1* **where** *ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)*
by (*cases ss1, auto*)
obtain *cfg2 ibT2 ibUT2 ls2* **where** *ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)*
by (*cases ss2, auto*)
note *ss = ss3 ss4 ss1 ss2*

obtain *pc1 vs1 avst1 h1 p1* **where**
cfg1: cfg1 = Config pc1 (State (Vstore vs1) avst1 h1 p1)
by (*cases cfg1*) (*metis state.collapse vstore.collapse*)
obtain *pc2 vs2 avst2 h2 p2* **where**
cfg2: cfg2 = Config pc2 (State (Vstore vs2) avst2 h2 p2)
by (*cases cfg2*) (*metis state.collapse vstore.collapse*)
obtain *pc3 vs3 avst3 h3 p3* **where**
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (*cases cfg3*) (*metis state.collapse vstore.collapse*)
obtain *pc4 vs4 avst4 h4 p4* **where**
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (*cases cfg4*) (*metis state.collapse vstore.collapse*)
note *cfg = cfg1 cfg2 cfg3 cfg4*

obtain *hh3* **where** *h3: h3 = Heap hh3* **by**(*cases h3, auto*)
obtain *hh4* **where** *h4: h4 = Heap hh4* **by**(*cases h4, auto*)
note *hh = h3 h4*

have *f1: ¬finalN ss1*
using $\Delta 1$ *finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB $\Delta 1$ -defs*
by *simp*

have *f2: ¬finalN ss2*
using $\Delta 1$ *finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB $\Delta 1$ -defs*
by *simp*

have *f3: ¬finalS ss3*
using $\Delta 1$ **unfolding** *ss apply-apply(frule $\Delta 1$ -implies)*
using *finalS-cond* **by** *simp*

have *f4: ¬finalS ss4*
using $\Delta 1$ **unfolding** *ss apply-apply(frule $\Delta 1$ -implies)*
using *finalS-cond* **by** *simp*

note *finals = f1 f2 f3 f4*

show *finalS ss3 = finalS ss4 \wedge finalN ss1 = finalS ss3 \wedge finalN ss2 = finalS ss4*
using *finals* **by** *auto*

then show *isIntO ss3 = isIntO ss4* **by** *simp*

```

show react (oor3  $\Delta 1$   $\Delta 2$   $\Delta e$ ) ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

show match1 (oor3  $\Delta 1$   $\Delta 2$   $\Delta e$ ) ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor3  $\Delta 1$   $\Delta 2$   $\Delta e$ ) ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor3  $\Delta 1$   $\Delta 2$   $\Delta e$ ) ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  show eqSec ss1 ss3
  using v sa  $\Delta 1$  unfolding ss
  by (simp add:  $\Delta 1$ -defs eqSec-def)

  show eqSec ss2 ss4
  using v sa  $\Delta 1$  unfolding ss
  by (simp add:  $\Delta 1$ -defs eqSec-def)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 1$  unfolding ss Van.eqAct-def
  by (simp-all add:  $\Delta 1$ -defs)

  show match12-12 (oor3  $\Delta 1$   $\Delta 2$   $\Delta e$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
  conjI impI)
    show validTransV (ss1, nextN ss1)
      by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
      by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff

```

```

show sstatO' statO ss1 ss2 = Diff
using v sa Δ1 sstat unfolding ss cfg statA'
apply(simp add: Δ1-defs sstatO'-def sstatA'-def)
using cases-6[of pc3] apply(elim disjE)
defer 1 defer 1
subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss status.distinct(1) newStat.simps by auto
  by simp-all
} note stat = this
have cfs:cfgs3 = [] cfs4 = [] using Δ1 unfolding ss apply-by (frule
Δ1-implies, auto)+

show (oor3 Δ1 Δ2 Δe) ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case spec-normal
  then show ?thesis using sa Δ1 stat unfolding ss by (simp add: Δ1-defs)

next
  case spec-mispred
  then show ?thesis using sa Δ1 stat unfolding ss by (simp add: Δ1-defs)

next
  case spec-Fence
  then show ?thesis using sa Δ1 stat unfolding ss by (simp add: Δ1-defs)

next
  case spec-resolve
  then show ?thesis using sa Δ1 stat unfolding ss by (simp add: Δ1-defs)

next
  case nonspec-mispred note nm3 = nonspec-mispred
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

    case nonspec-normal
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-normal
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:

```

```

Δ1-defs)
  next
    case spec-mispred
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-Fence
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-resolve
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case nonspec-mispred note nm4 = nonspec-mispred
    then show ?thesis
    using sa Δ1 stat v3 v4 nm3 nm4 unfolding ss cfg hh apply clarsimp
    using cases-6[of pc3] apply(elim disjE, simp-all add: Δ1-defs)
    by(rule oor3I2, simp add: Δ1-defs Δ2-defs, metis)
  qed(simp-all add: cfgs)
  next
    case nonspec-normal note nn3 = nonspec-normal
    show ?thesis using v4 [unfolded ss, simplified] proof(cases rule: stepS-cases)

      case nonspec-mispred
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-normal
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-mispred
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-Fence
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case spec-resolve
      then show ?thesis using sa Δ1 stat nn3 unfolding ss by (simp add:
Δ1-defs)
    next
      case nonspec-normal
      then show ?thesis using sa Δ1 stat v3 v4 nn3 unfolding ss cfg hh
apply clarsimp
    using cases-6[of pc3] apply(elim disjE)
    subgoal by (simp add: Δ1-defs)

```

```

      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal apply(rule oor3I1) by(simp add: $\Delta 1$ -defs, metis)
      subgoal apply(rule oor3I1) by (simp add:  $\Delta 1$ -defs, metis)
      subgoal apply(rule oor3I1) by (simp add:  $\Delta 1$ -defs, metis)
      subgoal apply(rule oor3I1) by (simp add:  $\Delta 1$ -defs, metis)
      apply(rule oor3I3) by (simp-all add:  $\Delta 1$ -defs  $\Delta e$ -defs)
    qed(simp-all add: cfgs)
  qed(simp-all add: cfgs)
qed
qed
qed
qed

```

lemma *step2: unwindIntoCond $\Delta 2$ $\Delta 1$*

proof(rule *unwindIntoCond-simpleI*)

fix *n ss3 ss4 statA ss1 ss2 statO*

assume *r: reachO ss3 reachO ss4 reachV ss1 reachV ss2*

and $\Delta 2$: $\Delta 2$ *n ss3 ss4 statA ss1 ss2 statO*

obtain *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3* **where** *ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)*

by (*cases ss3, auto*)

obtain *pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4* **where** *ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)*

by (*cases ss4, auto*)

obtain *cfg1 ibT1 ibUT1 ls1* **where** *ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)*

by (*cases ss1, auto*)

obtain *cfg2 ibT2 ibUT2 ls2* **where** *ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)*

by (*cases ss2, auto*)

note *ss = ss3 ss4 ss1 ss2*

obtain *pc3 vs3 avst3 h3 p3* **where**

lcfgs3: last cfgs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)

by (*cases last cfgs3*) (*metis state.collapse vstore.collapse*)

obtain *pc4 vs4 avst4 h4 p4* **where**

lcfgs4: last cfgs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)

by (*cases last cfgs4*) (*metis state.collapse vstore.collapse*)

note *lcfgs = lcfgs3 lcfgs4*

have *f1: \neg finalN ss1*

using $\Delta 2$ *finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB $\Delta 2$ -defs*

by *simp*

have *f2: \neg finalN ss2*

using $\Delta 2$ *finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB $\Delta 2$ -defs*

by *auto*

have *f3: \neg finalS ss3*

```

using  $\Delta 2$  unfolding ss apply-apply(frul  $\Delta 2$ -implies)
using finalS-cond-spec by simp

have f4:¬finalS ss4
  using  $\Delta 2$  unfolding ss apply-apply(frul  $\Delta 2$ -implies)
  using finalS-cond-spec by simp

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI,rule disjI1, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
  note hh = h3 h4

  show ¬ isSecO ss3
  using v sa  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

  show ¬ isSecO ss4
  using v sa  $\Delta 2$  unfolding ss apply clarsimp
  by (simp add:  $\Delta 2$ -defs, linarith)

```

```

show stat: statA = statA' ∨ statO = Diff
using v sa Δ2
apply (cases ss3, cases ss4, cases ss1, cases ss2)
apply(cases ss3', cases ss4', clarsimp)
unfolding ss statA' apply clarsimp
apply(simp-all add: Δ2-defs sstatA'-def)
apply(cases statO, simp-all) apply(cases statA, simp-all)
unfolding finalS-defs
by (smt (verit, ccfv-SIG) newStat.simps(1))

  have isO:is-Output (prog ! pcOf (last cfigs3)) is-Output (prog ! pcOf (last
cfigs4)) using Δ2-implies[OF Δ2[unfolded ss]] by auto
  have pstate3:pstate3 = pstate4 using Δ2[unfolded ss Δ2-defs] by fast

show Δ1 ∞ ss3' ss4' statA' ss1 ss2 statO

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sa stat Δ2 unfolding ss by (simp add: Δ2-defs)
next
  case nonspec-mispred
  then show ?thesis using sa stat Δ2 unfolding ss by (simp add: Δ2-defs)
next
  case spec-normal
  then show ?thesis using sa stat Δ2 v3 unfolding ss apply-
  apply(frule Δ2-implies) by(simp add: Δ2-defs)
next
  case spec-mispred
  then show ?thesis using sa stat Δ2 unfolding ss apply-
  apply(frule Δ2-implies) by (simp add: Δ2-defs)
next
  case spec-Fence
  then show ?thesis using sa stat Δ2 unfolding ss apply-
  apply(frule Δ2-implies) by (simp add: Δ2-defs)
next
  case spec-resolveI
  then show ?thesis using sa stat Δ2 unfolding ss apply-
  apply(frule Δ2-implies) by (simp add: Δ2-defs)
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfigs4) using
Δ2-implies[OF Δ2[unfolded ss]] unfolding pstate3 by auto
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sa stat Δ2 sr3 unfolding ss by (simp add:
Δ2-defs)
  next
  case nonspec-mispred
  then show ?thesis using sa stat Δ2 sr3 unfolding ss by (simp add:

```

```

Δ2-defs)
  next
  case spec-normal
  then show ?thesis using sa stat Δ2 sr3 unfolding ss by (simp add:
Δ2-defs)
  next
  case spec-mispred
  then show ?thesis using sa stat Δ2 sr3 unfolding ss by (simp add:
Δ2-defs)
  next
  case spec-Fence
  then show ?thesis using sa stat Δ2 sr3 unfolding ss by (simp add:
Δ2-defs)
  next
  case spec-resolveI
  then show ?thesis using sa stat Δ2 unfolding ss apply-
  apply(frul Δ2-implies) by (simp add: Δ2-defs)
  next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using sa stat Δ2 v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  by(frul Δ2-implies, simp add: Δ2-defs Δ1-defs, metis)
  next
  case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
  show ?thesis using sa stat Δ2 v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply(frul Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
  qed
  next
  case spec-resolveO note srO3 = spec-resolveO
  have cfigs4:cfigs4 ≠ [] using sa stat Δ2 unfolding ss apply-by(frul
Δ2-implies, auto)
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis by (simp add: cfigs4)
  next
  case nonspec-mispred
  then show ?thesis by (simp add: cfigs4)
  next
  case spec-normal
  then show ?thesis by (simp add: isO)
  next
  case spec-mispred
  then show ?thesis using isO by blast
  next
  case spec-Fence
  then show ?thesis using isO by blast
  next
  case spec-resolveI

```

```

    then show ?thesis using isO by blast
  next
    case spec-resolveO note srO4 = spec-resolveO
    show ?thesis using sa stat Δ2 v3 v4 srO3 srO4
    unfolding ss lcfgs hh apply-
    apply(frule Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
  next
    case spec-resolve note sr4 = spec-resolve(1,3-) isO(2)
    show ?thesis using sa stat Δ2 v3 v4 srO3 sr4
    unfolding ss lcfgs hh apply-
    apply(frule Δ2-implies) by (simp add: Δ2-defs Δ1-defs, metis)
  qed
qed
qed
qed
qed

```

```

lemma xx-le-NN[simp]:cfg = Config pc (State (Vstore vs) avst h p) ⇒ vs xx =
0 ⇒ vs xx < int NN
  using NN by auto

```

```

lemma match12I:match12 (oor3 Δ1' Δ3' Δe) ss3 ss4 statA ss1 ss2 statO ⇒
(∃ v < n. proact (oor3 Δ1' Δ3' Δe) v ss3 ss4 statA ss1 ss2 statO) ∨
  react (oor3 Δ1' Δ3' Δe) ss3 ss4 statA ss1 ss2 statO
apply(rule disjI2) unfolding react-def match1-def match2-def
by(simp-all add: finalS-def final-def)

```

```

lemma step1': unwindIntoCond Δ1' (oor3 Δ1' Δ3' Δe)
proof(rule unwindIntoCond-simpleIB)
  fix n ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and Δ1': Δ1' n ss3 ss4 statA ss1 ss2 statO

```

```

  obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

```

```

  obtain pc1 vs1 avst1 h1 p1 where

```

```

    cfg1: cfg1 = Config pc1 (State (Vstore vs1) avst1 h1 p1)
    by (cases cfg1) (metis state.collapse vstore.collapse)
obtain pc2 vs2 avst2 h2 p2 where
    cfg2: cfg2 = Config pc2 (State (Vstore vs2) avst2 h2 p2)
    by (cases cfg2) (metis state.collapse vstore.collapse)
obtain pc3 vs3 avst3 h3 p3 where
    cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
    by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
    cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
    by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

obtain hh1 where h1: h1 = Heap hh1 by(cases h1, auto)
obtain hh2 where h2: h2 = Heap hh2 by(cases h2, auto)
obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
note hh = h3 h4

have f1:¬finalN ss1
  using Δ1'
  unfolding ss apply–apply(frule Δ1'-implies)
  unfolding finalN-iff-finalB Δ1'-defs
  using finalB-pcOf-iff by simp

have f2:¬finalN ss2
  using Δ1'
  unfolding ss apply–apply(frule Δ1'-implies)
  unfolding finalN-iff-finalB Δ1'-defs
  using finalB-pcOf-iff by simp

have f3:¬finalS ss3
  using Δ1' unfolding ss apply–apply(frule Δ1'-implies)
  using finalS-cond by (simp add: Δ1'-defs)

have f4:¬finalS ss4
  using Δ1' unfolding ss apply–apply(frule Δ1'-implies)
  using finalS-cond by (simp add: Δ1'-defs)

note finals = f1 f2 f3 f4

show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalN ss2 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

have cfs:cfgs3 = [] cfgs4 = [] using Δ1'-implies[OF Δ1'[unfolded ss]] by auto

```

```

show ( $\exists v < n$ . proact (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) v ss3 ss4 statA ss1 ss2 statO)  $\vee$ 
  react (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) ss3 ss4 statA ss1 ss2 statO
using cases-6[of pcOf cfg1] apply(elim disjE)
subgoal using  $\Delta 1'$  unfolding ss by (simp add:  $\Delta 1'$ -defs, linarith)
subgoal using  $\Delta 1'$  unfolding ss by (simp add:  $\Delta 1'$ -defs, linarith)
subgoal proof(rule match12I, rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4 and pc:pcOf cfg1 = 2
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfgs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfgs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

show eqSec ss1 ss3
  using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
  by (metis not-gr-zero not-numeral-le-zero zero-less-numeral)

show eqSec ss2 ss4
  using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
  by (metis not-gr-zero not-numeral-le-zero zero-neq-numeral)

show Van.eqAct ss1 ss2
  using v sa  $\Delta 1'$  unfolding ss Van.eqAct-def
  apply (simp-all add:  $\Delta 1'$ -defs)
  by (metis  $\Delta 1'$   $\Delta 1'$ -implies ss)

show match12-12 (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2],unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  have cfgs4:cfgs4 = [] using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -defs by (clarify, metis
list.map-disc-iff)

  have notJump: $\neg$ is-IfJump (prog ! pcOf cfg3) using  $\Delta 1'$  pc unfolding ss
 $\Delta 1'$ -defs

```

```

by(simp add:  $\Delta 1'$ -defs sstatO'-def sstatA'-def)

{assume sstat: statA' = Diff
show sstatO' statO ss1 ss2 = Diff
using v sa  $\Delta 1'$  sstat pc unfolding ss cfg statA'
apply(simp add:  $\Delta 1'$ -defs sstatO'-def sstatA'-def)
apply(cases statO, simp-all) apply(cases statA, simp-all)
  using cfg finals ss by simp
} note stat = this

have pc4:pc4 = 2
  using v sa  $\Delta 1'$  pc unfolding ss cfg
  by (simp-all add:  $\Delta 1'$ -defs)

show (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ )  $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2)
(sstatO' statO ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case spec-normal
  then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-Fence
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-resolve
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case nonspec-mispred
    then show ?thesis using notJump by auto
  next
  case nonspec-normal note nn3 = nonspec-normal
show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

  case nonspec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-normal
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next

```

```

      case spec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-Fence
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-resolve
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case nonspec-normal note nn4 = nonspec-normal
    show ?thesis apply(rule oor3I1)
      using sa  $\Delta 1'$  stat pc pc4 v3 v4 nn3 config.sel(2) state.sel(2)
      unfolding ss cfg cfg1 cfg2 hh apply(simp add: $\Delta 1'$ -defs)
      using numeral-le-iff semiring-norm(69,72) by force
    qed(simp-all add: cfs)
    qed(simp-all add: cfs)
  qed
  qed
  subgoal proof(rule match12I, rule match12-simpleI, rule disjI2, intro conjI)
    fix ss3' ss4' stata'
    assume stata': stata' = sstata' stata ss3 ss4
      and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
      and sa: Opt.eqAct ss3 ss4 and pc:pcOf cfg1 = 3
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

  show eqSec ss1 ss3
    using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
    by (metis not-gr-zero not-numeral-le-zero zero-less-numeral)

  show eqSec ss2 ss4
    using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
    by (metis not-gr-zero not-numeral-le-zero zero-neq-numeral)

  show Van.eqAct ss1 ss2
    using v sa  $\Delta 1'$  unfolding ss Van.eqAct-def
    apply (simp-all add:  $\Delta 1'$ -defs)
    by (metis  $\Delta 1'$   $\Delta 1'$ -implies ss)

```

```

show match12-12 (oor3  $\Delta 1' \Delta 3' \Delta e$ ) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2],unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  have cfs4:cfs4 = [] using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -defs by (clarify,metis
map-is-Nil-conv)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
  using v sa  $\Delta 1'$  sstat pc unfolding ss cfg statA'
  apply(simp add:  $\Delta 1'$ -defs sstatO'-def sstatA'-def)
  apply(cases statO, simp-all) apply(cases statA, simp-all)
    using cfg finals ss by simp
  } note stat = this

  have pc4:pc4 = 3
    using v sa  $\Delta 1'$  pc unfolding ss cfg
    by (simp-all add:  $\Delta 1'$ -defs)

  show (oor3  $\Delta 1' \Delta 3' \Delta e$ )  $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2)
(ssstatO' statO ss1 ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case spec-normal
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-Fence
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case spec-resolve
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
  case nonspec-mispred note nm3 = nonspec-mispred
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

```

```

      case spec-normal
    then show ?thesis using sa  $\Delta 1'$  stat nm3 unfolding ss by (simp add:
 $\Delta 1'$ -defs cfigs4)
    next
      case spec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat nm3 unfolding ss by (simp add:
 $\Delta 1'$ -defs cfigs4)
    next
      case spec-Fence
    then show ?thesis using sa  $\Delta 1'$  stat nm3 unfolding ss by (simp add:
 $\Delta 1'$ -defs cfigs4)
    next
      case spec-resolve
    then show ?thesis using sa  $\Delta 1'$  stat nm3 unfolding ss by (simp add:
 $\Delta 1'$ -defs cfigs4)
    next
      case nonspec-normal
    then show ?thesis using sa  $\Delta 1'$  stat nm3 unfolding ss by (simp add:
 $\Delta 1'$ -defs cfigs4)
    next
      case nonspec-mispred note nm4 = nonspec-mispred
    show ?thesis apply(rule oor3I2)
      using sa pc4  $\Delta 1'$  stat pc v3 v4 nm3 nm4 config.sel(2) state.sel(2)
      unfolding ss cfg cfg1 cfg2 hh apply(simp add: $\Delta 1'$ -defs  $\Delta 3'$ -defs)
      by (metis empty-subsetI nat-less-le nat-neq-iff numeral-eq-iff semir-
ing-norm(89) set-eq-subset)
    qed(simp-all add: cfigs)
    next
      case nonspec-normal note nn3 = nonspec-normal
    show ?thesis using v4 [unfolded ss, simplified] proof(cases rule: stepS-cases)

      case nonspec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-normal
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-mispred
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-Fence
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-resolve
    then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:

```

```

 $\Delta 1'$ -defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  show ?thesis apply(rule oor3I1)
    using sa pc4  $\Delta 1'$  stat pc v3 v4 nm3 config.sel(2) state.sel(2)
    unfolding ss cfg cfg1 cfg2 hh apply(simp add: $\Delta 1'$ -defs)
    by (metis nat-le-linear nat-less-le numeral-eq-iff semiring-norm(88))
  qed(simp-all add: cfgs)
  qed(simp-all add: cfgs)
  qed
  subgoal apply(rule disjI1, rule exI[of - 2], rule conjI)
  subgoal using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -defs apply clarify
    apply(erule disjE)
  subgoal premises p using p(1,47) unfolding endPC by simp
  subgoal using enat-ord-simps(4) numeral-ne-infinity by presburger .
  unfolding proact-def proof(intro disjI2, intro conjI)
  assume pc:pcOf cfg1 = 4

  show  $\neg$  isSecV ss1 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg by auto

  show  $\neg$  isSecV ss2 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg by auto

  show Van.eqAct ss1 ss2 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg Van.eqAct-def
  by auto

  show move-12 (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) 2 ss3 ss4 statA ss1 ss2 statO
  unfolding move-12-def Let-def
  proof (rule exI[of - nextN ss1], rule exI[of - nextN ss2], intro conjI)
  show validTransV (ss1, nextN ss1)
    using  $\Delta 1'$  pc unfolding validTransV-iff-nextN ss  $\Delta 1'$ -defs
    by simp

  show validTransV (ss2, nextN ss2)
    using  $\Delta 1'$  pc unfolding validTransV-iff-nextN ss  $\Delta 1'$ -defs
    by simp
  have a1-0:array-loc aa1 0 avst3 = array-loc aa1 0 avst4
    using  $\Delta 1'$  pc unfolding cfg cfg1 ss  $\Delta 1'$ -defs array-loc-def by simp
  have pc1:pc1 = 4 using  $\Delta 1'$  pc unfolding cfg cfg1 ss  $\Delta 1'$ -defs by simp

  show oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$  2 ss3 ss4 statA (nextN ss1) (nextN ss2) (sstatO'
  statO ss1 ss2)
    apply(rule oor3I1)
    using  $\Delta 1'$  pc unfolding ss cfg cfg1 cfg2 hh h1 h2 endPC apply(simp
  add:  $\Delta 1'$ -defs)
    apply-apply(intro conjI)
    subgoal by (metis numeral-eq-enat)
    subgoal by (metis Nil-is-map-conv)
    subgoal by metis

```

```

    subgoal by metis
    subgoal unfolding sstatO'-def by simp
    subgoal using a1-0 by force
    subgoal unfolding a1-0 dist-def pc1 array-loc-def by simp
    subgoal by blast
    subgoal by (simp add: subset-insertI2)
    subgoal by (simp add: subset-insertI2) .
  qed
  qed
  subgoal apply(rule disjI1, rule exI[of - 1], rule conjI)
  subgoal using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -defs apply clarify
    apply(erule disjE)
    subgoal premises p using p(1,47) unfolding endPC by (simp add:
one-enat-def)
    subgoal by (metis enat-ord-code(4) one-enat-def) .
    unfolding proact-def proof(intro disjI2, intro conjI)
    assume pc:pcOf cfg1 = 5

    show  $\neg$  isSecV ss1 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg by auto

    show  $\neg$  isSecV ss2 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg by auto

    show Van.eqAct ss1 ss2 using  $\Delta 1'$  pc unfolding  $\Delta 1'$ -defs ss cfg Van.eqAct-def
  by auto

  show move-12 (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) 1 ss3 ss4 statA ss1 ss2 statO
    unfolding move-12-def Let-def
  proof (rule exI[of - nextN ss1], rule exI[of - nextN ss2], intro conjI)
    show validTransV (ss1, nextN ss1)
      using  $\Delta 1'$  pc unfolding validTransV-iff-nextN ss  $\Delta 1'$ -defs
      by simp

    show validTransV (ss2, nextN ss2)
      using  $\Delta 1'$  pc unfolding validTransV-iff-nextN ss  $\Delta 1'$ -defs
      by simp

    show oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$  1 ss3 ss4 statA (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)
      apply(rule oor3I1)
      using  $\Delta 1'$  pc unfolding ss cfg cfg1 cfg2 hh h1 h2 endPC apply(simp
add:  $\Delta 1'$ -defs)
      apply-apply(intro conjI)
      subgoal by (metis One-nat-def one-enat-def)
      subgoal by (metis Nil-is-map-conv)
      subgoal by metis
      subgoal by metis
      subgoal unfolding sstatO'-def by simp
      subgoal by (metis Suc-n-not-le-n eval-nat-numeral(3) nat-le-linear)
      subgoal by (metis atThenOutput-def insert-compr less-or-eq-imp-le

```

```

mult.commute nat-numeral pc subset-insertI2)
  subgoal by (simp add: subset-insertI2) .
  qed
  qed
subgoal proof(rule match12I, rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4 and pc:pcOf cfg1 = 6
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  show eqSec ss1 ss3
  using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
  by (metis not-gr-zero not-numeral-le-zero zero-less-numeral)

  show eqSec ss2 ss4
  using v sa  $\Delta 1'$  unfolding ss apply (simp add:  $\Delta 1'$ -defs)
  by (metis not-gr-zero not-numeral-le-zero zero-neq-numeral)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 1'$  unfolding ss Van.eqAct-def
  apply (simp-all add:  $\Delta 1'$ -defs)
  by (metis  $\Delta 1'$   $\Delta 1'$ -implies ss)

  show match12-12 (oor3  $\Delta 1'$   $\Delta 3'$   $\Delta e$ ) ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
  conjI impI)
    show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  have cfs4':cfs4 = [] using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -defs by (clarify,metis
  map-is-Nil-conv)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
  using v sa  $\Delta 1'$  sstat pc unfolding ss cfg statA'
  apply(simp add:  $\Delta 1'$ -defs sstatO'-def sstatA'-def)

```

```

apply(cases statO, simp-all) apply(cases statA, simp-all)
using cfg finals ss apply (simp split: if-splits)
unfolding dist-def by blast
} note stat = this

have pc4:pc4 = 6
  using v sa Δ1' pc unfolding ss cfg
  by (simp-all add: Δ1'-defs)

have notJump:¬is-IfJump (prog ! pcOf cfg3) using Δ1' pc unfolding ss
Δ1'-defs
  by(simp add: Δ1'-defs sstatO'-def sstatA'-def)

show (oor3 Δ1' Δ3' Δe) ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2)
(sstatO' statO ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case spec-normal
  then show ?thesis using sa Δ1' stat unfolding ss by (simp add:
Δ1'-defs)
  next
  case spec-mispred
    then show ?thesis using sa Δ1' stat unfolding ss by (simp add:
Δ1'-defs)
  next
  case spec-Fence
    then show ?thesis using sa Δ1' stat unfolding ss by (simp add:
Δ1'-defs)
  next
  case spec-resolve
    then show ?thesis using sa Δ1' stat unfolding ss by (simp add:
Δ1'-defs)
  next
  case nonspec-mispred
    then show ?thesis using notJump by auto
  next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

  case nonspec-mispred
    then show ?thesis using sa Δ1' stat nn3 unfolding ss by (simp add:
Δ1'-defs)
  next
  case spec-normal
    then show ?thesis using sa Δ1' stat nn3 unfolding ss by (simp add:
Δ1'-defs)
  next
  case spec-mispred

```

```

      then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-resolve
      then show ?thesis using sa  $\Delta 1'$  stat nn3 unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case nonspec-normal note nn4 = nonspec-normal
      show ?thesis apply(rule oor3I3)
        using sa  $\Delta 1'$  stat pc pc4 v3 v4 nn3 config.sel(2) state.sel(2)
        unfolding ss cfg cfg1 cfg2 hh by(simp add: $\Delta 1'$ -defs  $\Delta e$ -defs)
      qed(simp-all add: cfgs)
      qed(simp-all add: cfgs)
    qed
  qed
  using  $\Delta 1'$  unfolding ss by(simp add: $\Delta 1'$ -defs)
qed

```

lemma *step3'*: *unwindIntoCond* $\Delta 3'$ (*oor* $\Delta 3'$ $\Delta 1'$)

proof(rule *unwindIntoCond-simpleI*)

fix n $ss3$ $ss4$ *statA* $ss1$ $ss2$ *statO*

assume r : *reachO* $ss3$ *reachO* $ss4$ *reachV* $ss1$ *reachV* $ss2$

and $\Delta 3'$: $\Delta 3'$ n $ss3$ $ss4$ *statA* $ss1$ $ss2$ *statO*

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3,$
 $ibT3, ibUT3, ls3)$

by (*cases* $ss3$, *auto*)

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4,$
 $ibT4, ibUT4, ls4)$

by (*cases* $ss4$, *auto*)

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (*cases* $ss1$, *auto*)

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (*cases* $ss2$, *auto*)

note $ss = ss3$ $ss4$ $ss1$ $ss2$

obtain $pc1$ $vs1$ $avst1$ $h1$ $p1$ **where**

$cfg1$: $cfg1 = Config$ $pc1$ (*State* (*Vstore* $vs1$) $avst1$ $h1$ $p1$)

by (*cases* $cfg1$) (*metis* *state.collapse* *vstore.collapse*)

obtain $pc2$ $vs2$ $avst2$ $h2$ $p2$ **where**

$cfg2$: $cfg2 = Config$ $pc2$ (*State* (*Vstore* $vs2$) $avst2$ $h2$ $p2$)

by (*cases* $cfg2$) (*metis* *state.collapse* *vstore.collapse*)

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$cfg3$: $cfg3 = Config$ $pc3$ (*State* (*Vstore* $vs3$) $avst3$ $h3$ $p3$)

```

  by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
  cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg1 cfg2 cfg3 cfg4

obtain lpc3 lvs3 lavst3 lh3 lp3 where
  lcfgs3: last cfgs3 = Config lpc3 (State (Vstore lvs3) lavst3 lh3 lp3)
  by (cases last cfgs3) (metis state.collapse vstore.collapse)
obtain lpc4 lvs4 lavst4 lh4 lp4 where
  lcfgs4: last cfgs4 = Config lpc4 (State (Vstore lvs4) lavst4 lh4 lp4)
  by (cases last cfgs4) (metis state.collapse vstore.collapse)
note lcfgs = lcfgs3 lcfgs4

obtain hh1 where h1: h1 = Heap hh1 by(cases h1, auto)
obtain hh2 where h2: h2 = Heap hh2 by(cases h2, auto)

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
obtain lhh3 where lh3: lh3 = Heap lhh3 by(cases lh3, auto)
obtain lhh4 where lh4: lh4 = Heap lhh4 by(cases lh4, auto)
note hh = h3 h4 lh3 lh4 h1 h2

define a1-3 where a1-3:a1-3 = array-loc aa1 0 avst3
define a1-4 where a1-4:a1-4 = array-loc aa1 0 avst4
define a2-3 where a2-3:a2-3 = array-loc aa2 (nat (lvs3 vv * 512)) avst3
define a2-4 where a2-4:a2-4 = array-loc aa2 (nat (lvs4 vv * 512)) avst4

have butlast:butlast cfgs4 = []
  using  $\Delta 3'$  unfolding ss apply (simp add:  $\Delta 3'$ -defs)
  by (metis length-1-butlast length-map)

have h3-eq:hh3 = lhh3
  using cfg lcfgs hh  $\Delta 3'$  unfolding  $\Delta 3'$ -defs ss apply clarify
  using config.sel(2) getHheap.simps heap.sel last-in-set
  by metis

have h4-eq:hh4 = lhh4
  using cfg lcfgs hh  $\Delta 3'$  unfolding  $\Delta 3'$ -defs ss apply clarify
  using config.sel(2) getHheap.simps heap.sel last-in-set
  by (metis map-is-Nil-conv)

have f1: $\neg$ finalN ss1
  using  $\Delta 3'$  finalB-pc-iff' unfolding ss finalN-iff-finalB  $\Delta 3'$ -defs
  by simp

have f2: $\neg$ finalN ss2

```

```

using  $\Delta 3'$  finalB-pc-iff' unfolding ss cfg finalN-iff-finalB  $\Delta 3'$ -defs
by simp

have f3: $\neg$ finalS ss3
using  $\Delta 3'$  unfolding ss apply-apply(frul  $\Delta 3'$ -implies)
using finalS-cond-spec apply (simp add:  $\Delta 3'$ -defs) by metis

have f4: $\neg$ finalS ss4
using  $\Delta 3'$  unfolding ss apply-apply(frul  $\Delta 3'$ -implies)
using finalS-cond-spec apply (simp add:  $\Delta 3'$ -defs)
by (metis length-map)

note finals = f1 f2 f3 f4

show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

have notIO: $\neg$ is-getInput (prog ! pcOf (last cfs4))
using is-Output-pcOf is-getUntrustedInput-pcOf
 $\Delta 3'$ [unfolded ss  $\Delta 3'$ -defs] by auto
have pc:pcOf (last cfs4) = pcOf (last cfs3) using  $\Delta 3'$ [unfolded ss  $\Delta 3'$ -def]
by auto
have pstate3:pstate3 = pstate4 using  $\Delta 3'$ [unfolded ss  $\Delta 3'$ -defs] by auto

show react (oor  $\Delta 3'$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO

unfolding react-def proof(intro conjI)

show match1 (oor  $\Delta 3'$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor  $\Delta 3'$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor  $\Delta 3'$   $\Delta 1'$ ) ss3 ss4 statA ss1 ss2 statO
using cases-thenBranch[of pcOf (last cfs3)]
apply(elim disjE)
subgoal using  $\Delta 3'$  unfolding ss lcfs  $\Delta 3'$ -defs
by (clarify, metis atLeastAtMost-iff inThenBranch-def lcfs3 le-antisym less-irrefl-nat
less-or-eq-imp-le startOfThenBranch-def)
subgoal
proof(rule match12-simpleI, rule disjI2, intro conjI)
fix ss3' ss4' statA'
assume statA': statA' = sstatA' statA ss3 ss4
and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
and sa: Opt.eqAct ss3 ss4
and pc:pcOf (last cfs3) = 4
note v3 = v(1) note v4 = v(2)

```

```

have pc2:pc2 = 4
  using  $\Delta 3'$  pc unfolding ss cfg unfolding  $\Delta 3'$ -defs
  apply clarify
  by (metis config.sel(1))

obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
note ss = ss ss3' ss4'

show eqSec ss1 ss3
  using v sa  $\Delta 3'$  unfolding ss by (simp add:  $\Delta 3'$ -defs)

show eqSec ss2 ss4
  using v sa  $\Delta 3'$  unfolding ss by (simp add:  $\Delta 3'$ -defs)

show Van.eqAct ss1 ss2
  using v sa  $\Delta 3'$  unfolding ss Van.eqAct-def
  by (simp add:  $\Delta 3'$ -defs lessI less-or-eq-imp-le numeral-3-eq-3 pc)

show match12-12 (oor  $\Delta 3'$   $\Delta 1'$ ) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
    show sstatO' statO ss1 ss2 = Diff
      using v sa  $\Delta 3'$  sstat unfolding ss cfg statA'
      apply(simp add:  $\Delta 3'$ -defs sstatO'-def sstatA'-def)
      apply(cases statO, simp-all) apply(cases statA, simp-all)
      by (smt (z3) Nil-is-map-conv cfg finals ss status.distinct(1) new-
Stat.simps(1))
    } note stat = this

  show oor  $\Delta 3'$   $\Delta 1'$   $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)
    using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-mispred
      then show ?thesis using sa  $\Delta 3'$  stat unfolding ss by (simp add:

```

```

Δ3'-defs)
  next
  case spec-mispred
  then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-Fence
  then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
  next
  case nonspec-normal
  then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfs4) cfs4 ≠ []
using Δ3'-implies[OF Δ3'[unfolded ss]] unfolding pstate3 by auto
  show ?thesis
  using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-mispred
  then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-Fence
  then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-normal
  then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
  next
  case spec-resolve note sr4 = spec-resolve
  then show ?thesis
  using Δ3' sr3 sr4 pc2 lcfgs h3-eq h4-eq hh stat a1-3 a1-4
  unfolding ss cfg
  apply simp
  apply(rule oorI2)
  apply (simp add: Δ3'-defs Δ1'-defs butlast )
  apply clarsimp
  unfolding array-loc-def by simp
  next

```

```

case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
then show ?thesis
  using  $\Delta 3'$  sr3 sr4 pc2 lcfs h3-eq h4-eq hh stat a1-3 a1-4
  unfolding ss cfg
  apply simp
  apply(rule oorI2)
  apply (simp add:  $\Delta 3'$ -defs  $\Delta 1'$ -defs butlast )
  apply clarsimp
  unfolding array-loc-def by simp
qed(simp-all add: notIO)
next
  case spec-resolveO note srO3 = spec-resolveO
    have isO:is-Output (prog ! pcOf (last cfs4)) using  $\Delta 3'$ -implies[OF
 $\Delta 3'$ [unfolded ss]] srO3(2) by auto
    show ?thesis using v4[unfolded ss, simplified] proof(cases rule:
stepS-cases)
      case nonspec-mispred
        then show ?thesis using sa  $\Delta 3'$  stat srO3 unfolding ss by (simp
add:  $\Delta 3'$ -defs)
        next
          case spec-mispred
            then show ?thesis using sa  $\Delta 3'$  stat srO3 unfolding ss by (simp
add:  $\Delta 3'$ -defs)
            next
              case spec-Fence
                then show ?thesis using sa  $\Delta 3'$  stat srO3 unfolding ss by (simp
add:  $\Delta 3'$ -defs)
                next
                  case nonspec-normal
                    then show ?thesis using sa  $\Delta 3'$  stat srO3 unfolding ss by (simp
add:  $\Delta 3'$ -defs)
                    next
                      case spec-normal
                        then show ?thesis using sa  $\Delta 3'$  stat srO3 unfolding ss by (simp
add:  $\Delta 3'$ -defs)
                        next
                          case spec-resolveO note srO4 = spec-resolveO
                            then show ?thesis
                              using  $\Delta 3'$  srO3 srO4 pc2 lcfs h3-eq h4-eq hh stat a1-3 a1-4
                              unfolding ss cfg
                              apply simp
                              apply(rule oorI2)
                              apply (simp add:  $\Delta 3'$ -defs  $\Delta 1'$ -defs butlast )
                              apply clarsimp .
                            next
                              case spec-resolve note srO4 = spec-resolve(1,3-) isO
                                then show ?thesis
                                  using  $\Delta 3'$  srO3 srO4 pc2 lcfs h3-eq h4-eq hh stat a1-3 a1-4
                                  unfolding ss cfg

```

```

    apply simp
    apply(rule oorI2)
    apply (simp add:  $\Delta 3'$ -defs  $\Delta 1'$ -defs butlast )
    by clarsimp
  qed(simp-all add: notIO)

next
case spec-normal note sn3 = spec-normal
show ?thesis
  using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-mispred
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-Fence
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-resolve
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-resolveO note srO3 = spec-resolveO
  show ?thesis using sa  $\Delta 3'$  stat srO3 sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-normal note sn4 = spec-normal
  then show ?thesis
    using  $\Delta 3'$  sn3 sn4 pc2 lcfgs h3-eq h4-eq hh stat a1-3 a1-4
    unfolding ss cfg
    apply simp
    apply(rule oorI1)
    apply (simp add:  $\Delta 3'$ -defs butlast )
    apply clarsimp apply(intro conjI)
    subgoal by (smt (z3) config.sel(2) last-in-set state.sel(1) vstore.sel)
    subgoal by (smt (z3) config.sel(2) last-in-set state.sel(1) vstore.sel)
    subgoal unfolding array-loc-def by simp .
  qed(simp-all add: notIO)
  qed(simp-all add: notIO pc)
qed
qed

```

```

subgoal proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    and pc:pcOf (last cfgs3) = 5
  note v3 = v(1) note v4 = v(2)

  have pc2:pc2 = 5
    using Δ3' Δ3'-implies pc unfolding ss cfg Δ3'-defs
    apply clarify by (smt (z3) config.sel(1))

  obtain pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfgs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
  obtain pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfgs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
  note ss = ss ss3' ss4'

  show eqSec ss1 ss3
    using v sa Δ3' unfolding ss by (simp add: Δ3'-defs pc)

  show eqSec ss2 ss4
    using v sa Δ3' unfolding ss by (simp add: Δ3'-defs pc)

  show Van.eqAct ss1 ss2
    using v sa Δ3' unfolding ss Van.eqAct-def
    by (simp add: Δ3'-defs pc)

  show match12-12 (oor Δ3' Δ1') ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
  conjI impI)
    show validTransV (ss1, nextN ss1)
      by (simp add: f1 nextN-stepN)

    show validTransV (ss2, nextN ss2)
      by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
    show sstatO' statO ss1 ss2 = Diff
      using v sa Δ3' sstat unfolding ss cfg statA'
      apply (simp add: Δ3'-defs sstatO'-def sstatA'-def)
      apply (cases statO, simp-all) apply (cases statA, simp-all)
      by (smt (z3) Nil-is-map-conv cfg f3 f4 ss status.distinct(1) new-
  Stat.simps(1))
  }

```

```

} note stat = this
have notO:¬is-Output (prog ! pcOf (last cfgs4)) using pc Δ3'-implies[OF
Δ3'[unfolded ss]] by auto
have oo:¬is-Output (prog ! 5) by blast
show oor Δ3' Δ1' ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)
using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case nonspec-mispred
then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
next
case spec-mispred
then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
next
case spec-Fence
then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
next
case nonspec-normal
then show ?thesis using sa Δ3' stat unfolding ss by (simp add:
Δ3'-defs)
next
case spec-resolve note sr3 = spec-resolve
then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) cfgs4 ≠ []
using Δ3'-implies[OF Δ3'[unfolded ss]] unfolding pstate3 by auto
show ?thesis
using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
case nonspec-mispred
then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
next
case spec-mispred
then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
next
case spec-Fence
then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
next
case spec-normal
then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
next
case nonspec-normal note nn4 = nonspec-normal
then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
next
case spec-resolve note sr4 = spec-resolve

```

```

then show ?thesis
  using  $\Delta 3'$  sr3 sr4 pc2 lcfs h3-eq h4-eq hh stat
  unfolding ss cfg a1-3 a1-4
  apply simp apply(rule oorI2)
  apply (simp add:  $\Delta 3'$ -defs  $\Delta 1'$ -defs butlast)
  by blast
next
case spec-resolveO note sr4 = spec-resolveO(1,3-)
then show ?thesis
  using  $\Delta 3'$  sr3 sr4 pc2 lcfs h3-eq h4-eq hh stat
  unfolding ss cfg a1-3 a1-4
  apply simp apply(rule oorI2)
  apply (simp add:  $\Delta 3'$ -defs  $\Delta 1'$ -defs butlast)
  by blast
qed(simp-all add: notIO notO)

next
case spec-normal note sn3 = spec-normal
show ?thesis
  using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-mispred
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-Fence
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-resolve
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case nonspec-normal note nn4 = nonspec-normal
  then show ?thesis using sa  $\Delta 3'$  stat sn3 unfolding ss by (simp add:
 $\Delta 3'$ -defs)
  next
  case spec-normal note sn4 = spec-normal
  then show ?thesis
    using  $\Delta 3'$  sn3 sn4 pc2 lcfs h3-eq h4-eq hh stat
    unfolding ss cfg a1-3 a1-4
    apply simp apply(rule oorI1)
    apply (simp add:  $\Delta 3'$ -defs butlast)
    apply clarsimp
    by (smt (z3) config.sel(2) last-in-set state.sel(1) vstore.sel)
  qed(simp-all add: notIO notO)

```

```

      qed(simp-all add: notIO notO pc oo)
    qed
  qed
subgoal proof(rule match12-simpleI, rule disjI1, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    and pc:pcOf (last cfgs3) = 6
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfgs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfgs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  show  $\neg$  isSecO ss3
    using v sa  $\Delta 3'$  unfolding ss by (simp add:  $\Delta 3'$ -defs)

  show  $\neg$  isSecO ss4
    using v sa  $\Delta 3'$  unfolding ss by (simp add:  $\Delta 3'$ -defs)

  show stat: statA = statA'  $\vee$  statO = Diff
    using v sa  $\Delta 3'$ 
    unfolding ss statA' sstatA'-def
    apply(simp-all add:  $\Delta 3'$ -defs)
    apply (cases statA, simp-all)
    by (smt (verit, best) Nil-is-map-conv f3 f4 ss newStat.simps(1))

  show oor  $\Delta 3' \Delta 1' \infty$  ss3' ss4' statA' ss1 ss2 statO
    using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-mispred
      then show ?thesis using sa  $\Delta 3'$  stat unfolding ss by (simp add:
 $\Delta 3'$ -defs)
    next
      case spec-mispred
      then show ?thesis using sa  $\Delta 3'$  stat unfolding ss by (simp add:
 $\Delta 3'$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa  $\Delta 3'$  stat unfolding ss by (simp add:
 $\Delta 3'$ -defs)
    next
      case nonspec-normal
      then show ?thesis using sa  $\Delta 3'$  stat unfolding ss by (simp add:

```

```

Δ3'-defs)
  next
    case spec-normal note sn3 = spec-normal
    show ?thesis using sa Δ3' stat sn3 pc v3 unfolding ss by (simp add:
Δ3'-defs)
  next

    case spec-resolve note sr3 = spec-resolve
    show ?thesis using v4[unfolded ss, simplified] proof(cases rule:
stepS-cases)
      case nonspec-mispred
      then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
    next
      case spec-mispred
      then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
    next
      case spec-Fence
      then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
    next
      case nonspec-normal
      then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
    next
      case spec-normal
      then show ?thesis using sa Δ3' stat sr3 unfolding ss by (simp add:
Δ3'-defs)
    next
      case spec-resolve note sr4 = spec-resolve
      then show ?thesis
        using Δ3' sr3 sr4 lcfgs hh stat a2-3 a2-4
          butlast array-locBase le-refl
          unfolding ss cfg
          apply simp
          apply(rule oorI2)
          apply (simp add: Δ3'-defs Δ1'-defs, intro conjI, metis)
          apply meson apply meson apply blast by meson
    next
      case spec-resolveO note sr4 = spec-resolveO
      then show ?thesis
        using Δ3' sr3 sr4 lcfgs hh stat a2-3 a2-4
          butlast array-locBase le-refl
          unfolding ss cfg
          apply simp
          apply(rule oorI2)
          apply (simp add: Δ3'-defs Δ1'-defs, intro conjI, metis)
          apply meson apply meson apply blast by meson

```

```

qed(simp-all add: notIO)
next

case spec-resolveO note srO3 = spec-resolveO
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule:
stepS-cases)
    case nonspec-mispred
      then show ?thesis using sa Δ3' stat srO3 unfolding ss by (simp
add: Δ3'-defs)
    next
      case spec-mispred
        then show ?thesis using sa Δ3' stat srO3 unfolding ss by (simp
add: Δ3'-defs)
      next
        case spec-Fence
          then show ?thesis using sa Δ3' stat srO3 unfolding ss by (simp
add: Δ3'-defs)
        next
          case nonspec-normal
            then show ?thesis using sa Δ3' stat srO3 unfolding ss by (simp
add: Δ3'-defs)
          next
            case spec-normal
              then show ?thesis using sa Δ3' stat srO3 unfolding ss by (simp
add: Δ3'-defs)
            next
              case spec-resolve note srO4 = spec-resolve
                then show ?thesis
                  using Δ3' srO3 srO4 lcfgs hh stat a2-3 a2-4
butlast array-locBase le-refl
                  unfolding ss cfg
                  apply simp
                  apply(rule oorI2)
                  apply (simp add: Δ3'-defs Δ1'-defs, intro conjI, metis)
                  apply meson apply meson apply blast by meson
                next
                  case spec-resolveO note sr4 = spec-resolveO
                    then show ?thesis
                      using Δ3' srO3 sr4 lcfgs hh stat a2-3 a2-4
butlast array-locBase le-refl
                      unfolding ss cfg
                      apply simp
                      apply(rule oorI2)
                      apply (simp add: Δ3'-defs Δ1'-defs, intro conjI, metis)
                      apply meson apply meson apply blast by meson
                    qed(simp-all add: notIO)
                qed(simp-all add: notIO pc)
              qed
            subgoal using Δ3' unfolding ss lcfgs Δ3'-defs

```

by (simp add: avstoreOf.cases elseBranch-def lcfgs3) .
qed
qed

lemma *stepe: unwindIntoCond* $\Delta e \Delta e$

proof(rule *unwindIntoCond-simpleI*)

fix $n \text{ ss3 ss4 statA ss1 ss2 statO}$

assume $r: \text{reachO ss3 reachO ss4 reachV ss1 reachV ss2}$

and $\Delta e: \Delta e n \text{ ss3 ss4 statA ss1 ss2 statO}$

obtain $pstate3 \text{ cfg3 cfgs3 ibT3 ibUT3 ls3}$ **where** $ss3: ss3 = (pstate3, \text{cfg3}, \text{cfgs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$

by (cases $ss3$, auto)

obtain $pstate4 \text{ cfg4 cfgs4 ibT4 ibUT4 ls4}$ **where** $ss4: ss4 = (pstate4, \text{cfg4}, \text{cfgs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$

by (cases $ss4$, auto)

obtain $\text{cfg1 ibT1 ibUT1 ls1}$ **where** $ss1: ss1 = (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$

by (cases $ss1$, auto)

obtain $\text{cfg2 ibT2 ibUT2 ls2}$ **where** $ss2: ss2 = (\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$

by (cases $ss2$, auto)

note $ss = ss3 \text{ ss4 ss1 ss2}$

obtain $pc3 \text{ vs3 avst3 h3 p3}$ **where**

$\text{cfg3}: \text{cfg3} = \text{Config } pc3 \text{ (State (Vstore vs3) avst3 h3 p3)}$

by (cases cfg3) (metis *state.collapse vstore.collapse*)

obtain $pc4 \text{ vs4 avst4 h4 p4}$ **where**

$\text{cfg4}: \text{cfg4} = \text{Config } pc4 \text{ (State (Vstore vs4) avst4 h4 p4)}$

by (cases cfg4) (metis *state.collapse vstore.collapse*)

note $\text{cfg} = \text{cfg3 } \text{cfg4}$

obtain $hh3$ **where** $h3: h3 = \text{Heap } hh3$ **by**(cases $h3$, auto)

obtain $hh4$ **where** $h4: h4 = \text{Heap } hh4$ **by**(cases $h4$, auto)

note $hh = h3 \text{ h4}$

show $\text{finalS } ss3 = \text{finalS } ss4 \wedge \text{finalN } ss1 = \text{finalS } ss3 \wedge \text{finalN } ss2 = \text{finalS } ss4$

using $\Delta e \text{ Opt.final-def Prog.endPC-def finalS-def stepS-endPC}$

unfolding $\Delta e\text{-defs } ss$ **by** *clarsimp*

then show $\text{isIntO } ss3 = \text{isIntO } ss4$ **by** *simp*

show $\text{react } \Delta e \text{ ss3 ss4 statA ss1 ss2 statO}$

unfolding *react-def* **proof**(intro *conjI*)

show $\text{match1 } \Delta e \text{ ss3 ss4 statA ss1 ss2 statO}$

unfolding *match1-def* **by** (simp add: *finalS-def final-def*)

show $\text{match2 } \Delta e \text{ ss3 ss4 statA ss1 ss2 statO}$

unfolding *match2-def* **by** (simp add: *finalS-def final-def*)

```

    show match12 Δe ss3 ss4 statA ss1 ss2 statO
      apply(rule match12-simpleI)
      using Δe stepS-endPC unfolding ss
      by (simp add: Δe-defs)
  qed
qed

lemmas theConds = step0 step1 step2
          step1' step3' stepe

proposition rsecure
proof -
  define m where m: m ≡ (6::nat)
  define Δs where Δs: Δs ≡ λi::nat.
    if i = 0 then Δ0
    else if i = 1 then Δ1
    else if i = 2 then Δ2
    else if i = 3 then Δ1'
    else if i = 4 then Δ3'
    else Δe
  define nxt where nxt: nxt ≡ λi::nat.
    if i = 0 then {0,1,3::nat}
    else if i = 1 then {1,2,5}
    else if i = 2 then {1}
    else if i = 3 then {3,4,5}
    else if i = 4 then {4,3}
    else {5}
  show ?thesis apply(rule distrib-unwind-rsecure[of m nxt Δs])
    subgoal unfolding m by auto
    subgoal unfolding nxt m by auto
    subgoal using init unfolding Δs by auto
    subgoal
      unfolding m nxt Δs apply (simp split: if-splits)
      using theConds
      unfolding oor-def oor3-def oor4-def by auto .
  qed
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 **by** *auto*

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 \neq *aa2* *aa1* \neq *vv* *aa1* \neq *xx* *aa1* \neq *temp* *aa1* \neq *tt*

aa2 \neq *aa1* *aa2* \neq *vv* *aa2* \neq *xx* *aa2* \neq *temp* *aa2* \neq *tt*

vv \neq *aa1* *vv* \neq *aa2* *vv* \neq *xx* *vv* \neq *temp* *vv* \neq *tt*

xx \neq *aa1* *xx* \neq *aa2* *xx* \neq *vv* *xx* \neq *temp* *xx* \neq *tt*

tt \neq *aa1* *tt* \neq *aa2* *tt* \neq *vv* *tt* \neq *temp* *tt* \neq *xx*

temp \neq *aa1* *temp* \neq *aa2* *temp* \neq *vv* *temp* \neq *xx* *temp* \neq *tt*

unfolding *vvars-defs* **by** *auto*

consts *size-aa1* :: *nat*

consts *size-aa2* :: *nat*

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

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

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

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

definition *prog* \equiv

[
~~/~~ *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 $PC \equiv \{0..11\}$

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 $cases-12: (i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee$
 $i = 6 \vee i = 7 \vee i = 8 \vee i = 9 \vee i = 10 \vee i = 11 \vee i = 12 \vee i > 12$

apply($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal **for** i **apply**($cases\ i, simp-all$)

subgoal for i apply(*cases i, simp-all*)
subgoal for i apply(*cases i, simp-all*)
subgoal for i apply(*cases i, simp-all*)

lemma *xx-0-cases: vs xx = 0 \vee vs xx \neq 0 by auto*

lemma *xx-NN-cases: vs xx < int NN \vee vs xx \geq int NN by auto*

lemma *is-IfJump-pcOf[simp]:*
pcOf cfg < 12 \implies is-IfJump (prog ! (pcOf cfg)) \longleftrightarrow pcOf cfg = 3 \vee pcOf cfg = 5
apply(*cases cfg*) **subgoal for** *pc s using cases-12[of pcOf cfg]*
by (*auto simp: prog-def*) .

lemma *is-IfJump-pc[simp]:*
pc < 12 \implies is-IfJump (prog ! pc) \longleftrightarrow pc = 3 \vee pc = 5
using *cases-12[of pc]*
by (*auto simp: prog-def*)

lemma *eq-Fence-pc[simp]:*
pc < 12 \implies prog ! pc = Fence \longleftrightarrow pc = 7
using *cases-12[of pc]*
by (*auto simp: prog-def*)

lemma *output1[simp]:*
prog ! 9 = Output U (V tt) by(simp add: prog-def)
lemma *output2[simp]:*
prog ! 11 = Output U (N 0) by(simp add: prog-def)
lemma *is-if[simp]:is-IfJump (prog ! 3) by(simp add: prog-def)*

lemma *is-nif1[simp]: \neg is-IfJump (prog ! 6) by(simp add: prog-def)*
lemma *is-nif2[simp]: \neg is-IfJump (prog ! 7) by(simp add: prog-def)*

lemma *is-nin1[simp]: \neg is-getInput (prog ! 6) by(simp add: prog-def)*
lemma *is-nout1[simp]: \neg is-Output (prog ! 6) by(simp add: prog-def)*
lemma *is-nin2[simp]: \neg is-getInput (prog ! 10) by(simp add: prog-def)*
lemma *is-nout2[simp]: \neg is-Output (prog ! 10) by(simp add: prog-def)*

lemma *fence[simp]:prog ! 7 = Fence by(simp add: prog-def)*

lemma *nfence[simp]:prog ! 6 \neq Fence by(simp add: prog-def)*

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*
by (*standard*, *simp add: prog-def*)

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** ⟨→*MM*⟩ 55)
where *x* →*MM* *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* = 12
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getInput-pcOf[simp]*: $pcOf\ cfg < 12 \implies is_getInput\ (prog!(pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 4$
using *cases-12[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *getUntrustedInput-pcOf[simp]*: $prog!4 = Input\ U\ xx$
by (*auto simp: prog-def*)

lemma *getInput-not6[simp]*: $\neg is_getInput\ (prog\ !\ 6)$ **by** (*auto simp: prog-def*)
lemma *getInput-not7[simp]*: $\neg is_getInput\ (prog\ !\ 7)$ **by** (*auto simp: prog-def*)
lemma *getInput-not10[simp]*: $\neg is_getInput\ (prog\ !\ 10)$ **by** (*auto simp: prog-def*)

lemma *is-Output-pcOf[simp]*: $pcOf\ cfg < 12 \implies is_Output\ (prog!(pcOf\ cfg)) \longleftrightarrow (pcOf\ cfg = 9 \vee pcOf\ cfg = 11)$
using *cases-12[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *is-Output*: $is_Output\ (prog\ !\ 9)$
unfolding *is-Output-def prog-def* **by** *auto*
lemma *is-Output-1*: $is_Output\ (prog\ !\ 11)$
unfolding *is-Output-def prog-def* **by** *auto*

lemma *isSecV-pcOf[simp]*:
 $isSecV\ (cfg,ibT,ibUT) \longleftrightarrow pcOf\ cfg = 0$
using *isSecV-def* **by** *simp*

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

lemma *getInputT-not[simp]*: $pcOf\ cfg < 12 \implies (prog\ !\ pcOf\ cfg) \neq Input\ T\ inp$
apply(*cases cfg*) **subgoal for** *pc s* **using** *cases-12[of pcOf cfg]* **by** (*auto simp: prog-def*) .

lemma *getActV-pcOf[simp]*:
 $pcOf\ cfg < 12 \implies getActV\ (cfg,ibT,ibUT,ls) = (if\ pcOf\ cfg = 4\ then\ lhd\ ibUT\ else\ \perp)$
apply(*subst getActV-simps*) **unfolding** *prog-def*
apply *simp*
using *getActV-simps*
using *cases-12[of pcOf cfg]*
by *auto*

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
 $)$
apply(*subst* *getObsV-simps*)
unfolding *prog-def* **apply** *simp*
using *getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def*
One-nat-def **by** *presburger*

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
 $)$
apply(*subst* *getActO-simps*)
apply(*cases* *cfgs, auto*)
unfolding *prog-def*
apply(*cases* $pcOf\ cfg = 4$, *auto*)
using *getActV-simps getActV-pcOf prog-def* **by** *simp*

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
 $)$
apply(*subst* *getObsO-simps*)
apply(*cases* *cfgs, auto*)
unfolding *prog-def*
using *getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def*
One-nat-def **by** *presburger*

lemma *eqSec-pcOf[simp]*:
 $eqSec\ (cfg1, ibT1, ibUT1, ls1)\ (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3) \longleftrightarrow$
 $(pcOf\ cfg1 = 0 \longleftrightarrow pcOf\ cfg3 = 0 \wedge cfgs3 = []) \wedge$
 $(pcOf\ cfg1 = 0 \longrightarrow stateOf\ cfg1 = stateOf\ cfg3)$
unfolding *eqSec-def* **by** *simp*

lemma *getActInput:pc4 = 4 \implies pc3 = 4 \implies cfgs3 = [] \implies cfgs4 = [] \implies*
 $getActO\ (pstate3, Config\ pc3\ (State\ (Vstore\ us3)\ avst3\ h3\ p3), [], ibT3, ibUT3,$
 $ls3) =$
 $getActO\ (pstate4, Config\ pc4\ (State\ (Vstore\ us4)\ avst4\ h4\ p4), [], ibT4, ibUT4,$
 $ls4)$

$\implies \text{lhs } \text{ibUT3} = \text{lhs } \text{ibUT4}$
using *getActO-pcOf zero-less-numeral* **by** *auto*

lemma *nextB-pc0[simp]*:
 $\text{nextB } (\text{Config } 0 \ s, \ \text{ibT}, \ \text{ibUT}) =$
 $(\text{Config } 1 \ s, \ \text{ibT}, \ \text{ibUT})$
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc0[simp]*:
 $\text{readLocs } (\text{Config } 0 \ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1[simp]*:
 $\text{nextB } (\text{Config } 1 \ (\text{State } (\text{Vstore } \text{vs}) \ \text{avst } \text{hh } \text{p}), \ \text{ibT}, \ \text{ibUT}) =$
 $((\text{Config } 2 \ (\text{State } (\text{Vstore } (\text{vs}(tt := 0))) \ \text{avst } \text{hh } \text{p})), \ \text{ibT}, \ \text{ibUT})$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1'[simp]*:
 $\text{nextB } (\text{Config } (\text{Suc } 0) \ (\text{State } (\text{Vstore } \text{vs}) \ \text{avst } \text{hh } \text{p}), \ \text{ibT}, \ \text{ibUT}) =$
 $((\text{Config } 2 \ (\text{State } (\text{Vstore } (\text{vs}(tt := 0))) \ \text{avst } \text{hh } \text{p})), \ \text{ibT}, \ \text{ibUT})$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1[simp]*:
 $\text{readLocs } (\text{Config } 1 \ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc1'[simp]*:
 $\text{readLocs } (\text{Config } (\text{Suc } 0) \ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc2[simp]*:
 $\text{nextB } (\text{Config } 2 \ (\text{State } (\text{Vstore } \text{vs}) \ \text{avst } \text{hh } \text{p}), \ \text{ibT}, \ \text{ibUT}) =$
 $((\text{Config } 3 \ (\text{State } (\text{Vstore } (\text{vs}(xx := 1))) \ \text{avst } \text{hh } \text{p})), \ \text{ibT}, \ \text{ibUT})$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc2[simp]*:
 $\text{readLocs } (\text{Config } 2 \ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

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

vs xx ≠ 0 \implies

nextB (*Config 3* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*) =
(*Config 4* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*)

apply(*subst nextB-IfTrue*)

unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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*)

apply(*subst nextB-IfFalse*)

unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

lemma *nextB-pc3*:

nextB (*Config 3* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*) =

(*Config* (*if vs xx ≠ 0 then 4 else 11*) (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*)

by(*cases vs xx = 0, auto*)

lemma *readLocs-pc3[simp]*:

readLocs (*Config 3 s*) = {}

unfolding *endPC-def readLocs-def* **unfolding** *prog-def Eq-def* **by** *auto*

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*)

apply(*subst nextM-IfTrue*)

unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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

vs xx ≠ 0 \implies

nextM (*Config 3* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*) =
(*Config 11* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*)

apply(*subst nextM-IfFalse*)

unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

lemma *nextM-pc3*:

nextM (*Config 3* (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*) =

(*Config* (*if vs xx ≠ 0 then 11 else 4*) (*State* (*Vstore vs*) *avst hh p*), *ibT,ibUT*)

by(*cases vs xx = 0, auto*)

lemma *nextB-pc4[simp]*:

ibUT ≠ 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*)

apply(subst nextB-getUntrustedInput')
unfolding endPC-def **unfolding** prog-def **by** auto

lemma readLocs-pc4[simp]:
readLocs (Config 4 s) = {}
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

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)
apply(subst nextB-IfTrue)
unfolding endPC-def **unfolding** prog-def Eq-def **by** auto

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)
apply(subst nextB-IfFalse)
unfolding endPC-def **unfolding** prog-def Eq-def **by** auto

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)
by(cases vs xx < NN, auto)

lemma readLocs-pc5[simp]:
readLocs (Config 5 s) = {}
unfolding endPC-def readLocs-def **unfolding** prog-def Eq-def **by** auto

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)
apply(subst nextM-IfTrue)
unfolding endPC-def **unfolding** prog-def Eq-def **by** auto

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)
apply(subst nextM-IfFalse)
unfolding endPC-def **unfolding** prog-def Eq-def **by** auto

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)

by(*cases vs xx < NN, auto*)

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc6[simp]*:

readLocs (*Config 6* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa1* (*nat* (*vs xx*))
avst}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc7[simp]*:

nextB (*Config 7 s, ibT,ibUT*) = (*Config 8 s, ibT,ibUT*)
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc7[simp]*:

readLocs (*Config 7 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc8[simp]*:

readLocs (*Config 8* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa2* (*nat* (*vs vv * SS*))
avst}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc9[simp]*:

nextB (*Config 9 s, ibT,ibUT*) = (*Config 10 s, ibT,ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc9*[simp]:
readLocs (*Config 9 s*) = {}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc10*[simp]:
nextB (*Config 10 s, ibT, ibUT*) = (*Config 3 s, ibT, ibUT*)
apply(*subst nextB-Jump*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc10*[simp]:
readLocs (*Config 10 s*) = {}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc11*[simp]:
nextB (*Config 11 s, ibT, ibUT*) =
(*Config 12 s, ibT, ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc11*[simp]:
readLocs (*Config 11 s*) = {}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *map-L1:length cfgs = Suc 0* \implies
pcOf (*last cfgs*) = *y* \implies *map pcOf cfgs* = [*y*]
by (*smt (verit, del-insts) Suc-length-conv cfgs-map last.simps*
length-0-conv map-eq-Cons-conv nth-Cons-0 numeral-2-eq-2)

lemma *map-L2:length cfgs = 2* \implies
pcOf (*cfgs ! 0*) = *x* \implies
pcOf (*last cfgs*) = *y* \implies *map pcOf cfgs* = [*x, y*]
by (*smt (verit) Suc-length-conv cfgs-map last.simps*
length-0-conv map-eq-Cons-conv nth-Cons-0 numeral-2-eq-2)

lemma *length2-butlast:length cfgs = 2* \implies (*cfgs ! 0*) = *last (butlast cfgs)*
by (*cases cfgs, auto*)

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)*
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-12*[*of pc*] **apply** *safe*
subgoal apply *simp apply*(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply(cases vs $xx = 0$)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply(cases vs $xx = 0$)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply(cases vs $xx = 0$)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, metis llist.exhaust-sel)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, metis llist.exhaust-sel) .
subgoal apply(cases vs $xx < NN$)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def) .

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
apply (simp add: prog-def)
using nextB-pc5 prog-def by presburger
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal by simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def) . .

lemma not-finalB:

$pc < 12 \implies (pc = 4 \longrightarrow ibUT \neq LNil) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$
using nextB-stepB-pc by (simp add: stepB-iff-nextB)

lemma finalB-pc-iff':

$pc < 12 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 4 \wedge ibUT = LNil)$
subgoal apply safe
subgoal using nextB-stepB-pc[of pc] by (auto simp add: stepB-iff-nextB)
subgoal using nextB-stepB-pc[of pc] by (auto simp add: stepB-iff-nextB)
subgoal using finalB-iff getUntrustedInput-pcOf by auto . .

lemma finalB-pc-iff:

$pc \leq 12 \implies$
 $finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$
 $(pc = 12 \vee pc = 4 \wedge ibUT = LNil)$
using Prog.finalB-iff endPC finalB-pc-iff' order-le-less finalB-iff
by metis

lemma finalB-pcOf-iff[simp]:

$pcOf\ cfg \leq 12 \implies$
 $finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 12 \vee pcOf\ cfg = 4 \wedge ibUT = LNil)$

by (metis config.collapse finalB-pc-iff)

lemma *finalS-cond:pcOf* $cfg < 12 \implies noMisSpec\ cfgs \implies ibUT \neq LNil \implies \neg$
finalS (*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*)

apply(*cases* *cfg*)

subgoal for *pc s* **apply**(*cases* *s*)

subgoal for *vst avst hh p* **apply**(*cases* *vst*, *cases* *avst*, *cases* *hh*)

subgoal for *vs as h*

using *cases-12*[of *pc*] **apply**(*elim* *disjE*) **unfolding** *finalS-defs* *noMisSpec-def*

subgoal using *nonspec-normal*[of [] *Config pc* (*State* (*Vstore vs*) *avst hh p*)

pstate pstate ibT ibUT

Config 1 (*State* (*Vstore vs*) *avst hh p*)

ibT ibUT [] *ls* \cup *readLocs* (*Config pc* (*State* (*Vstore*

vs) *avst hh p*) *ls*]

using *is-IfJump-pc* **by** *force*

subgoal apply(*frule* *nonspec-normal*[of *cfgs* *Config pc* (*State* (*Vstore vs*) *avst*
hh p)

pstate pstate ibT ibUT

Config 2 (*State* (*Vstore* (*vs(tt:= 0)*)) *avst hh p*)

ibT ibUT [] *ls* \cup *readLocs* (*Config pc* (*State* (*Vstore*

vs) *avst hh p*) *ls*]

prefer 7 **subgoal by** *metis* **by** *simp-all*

subgoal apply(*frule* *nonspec-normal*[of *cfgs* *Config pc* (*State* (*Vstore vs*) *avst*
hh p)

pstate pstate ibT ibUT

Config 3 (*State* (*Vstore* (*vs(xx:= 1)*)) *avst hh p*)

ibT ibUT [] *ls* \cup *readLocs* (*Config pc* (*State* (*Vstore*

vs) *avst hh p*) *ls*]

prefer 7 **subgoal by** *metis* **by** *simp-all*

subgoal apply(*cases* *mispred pstate* [3])

subgoal apply(*frule* *nonspec-mispred*[of *cfgs* *Config pc* (*State* (*Vstore vs*) *avst*
hh p)

(*Vstore vs*) *avst hh p*)]

pstate update pstate [*pcOf* (*Config pc* (*State*

(*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 *Config* (*if vs xx* \neq 0 *then* 11 *else* 4)

(*State* (*Vstore vs*) *avst hh p*)

ibT ibUT [*Config* (*if vs xx* \neq 0 *then* 11 *else* 4)

avst hh p) *ls*]

ls \cup *readLocs* (*Config pc* (*State* (*Vstore vs*)

prefer 9 **subgoal by** *metis* **by** (*simp* *add: finalM-iff*)+

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*])

$$\begin{array}{l} pstate \ pstate \ ibT \ ibUT \\ Config \ (if \ vs \ xx \neq \ 0 \ then \ 4 \ else \ 11) \ (State \ (Vstore \ vs) \\ avst \ hh \ p) \\ ibT \ ibUT \ [] \ ls \cup \ readLocs \ (Config \ pc \ (State \ (Vstore \\ vs) \ avst \ hh \ p)) \ ls] \\ \mathbf{prefer \ 7 \ subgoal \ by \ metis \ by \ simp-all \ .} \end{array}$$

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*])

$$\begin{array}{l} pstate \ pstate \ ibT \ ibUT \\ Config \ 5 \ (State \ (Vstore \ (vs(xx:= \ lhd \ ibUT))) \ avst \ hh \\ p) \\ ibT \ ltl \ ibUT \ [] \ ls \cup \ readLocs \ (Config \ pc \ (State \ (Vstore \\ vs) \ avst \ hh \ p)) \ ls] \\ \mathbf{prefer \ 7 \ subgoal \ by \ metis \ by \ simp-all} \end{array}$$

subgoal apply(*cases mispred pstate [5]*)
subgoal apply(*frule nonspec-mispred*[of cfigs *Config pc (State (Vstore vs) avst hh p)*])

$$\begin{array}{l} pstate \ update \ pstate \ [pcOf \ (Config \ pc \ (State \\ (Vstore \ vs) \ avst \ hh \ p))] \\ ibT \ ibUT \ Config \ (if \ vs \ xx < NN \ then \ 6 \ else \\ 10) \ (State \ (Vstore \ vs) \ avst \ hh \ p) \\ ibT \ ibUT \ Config \ (if \ vs \ xx < NN \ then \ 10 \ else \\ 6) \ (State \ (Vstore \ vs) \ avst \ hh \ p) \\ ibT \ ibUT \ [Config \ (if \ vs \ xx < NN \ then \ 10 \ else \\ 6) \ (State \ (Vstore \ vs) \ avst \ hh \ p)] \\ ls \cup \ readLocs \ (Config \ pc \ (State \ (Vstore \ vs) \\ avst \ hh \ p)) \ ls] \\ \mathbf{prefer \ 9 \ subgoal \ by \ metis \ by \ (simp \ add: \ finalM-iff)+} \end{array}$$

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*])

$$\begin{array}{l} pstate \ pstate \ ibT \ ibUT \\ Config \ (if \ vs \ xx < NN \ then \ 6 \ else \ 10) \ (State \ (Vstore \\ vs) \ avst \ hh \ p) \\ ibT \ ibUT \ [] \ ls \cup \ readLocs \ (Config \ pc \ (State \ (Vstore \\ vs) \ avst \ hh \ p)) \ ls] \\ \mathbf{prefer \ 7 \ subgoal \ by \ metis \ by \ simp-all \ .} \end{array}$$

subgoal apply(*frule nonspec-normal*[of cfigs *Config pc (State (Vstore vs) avst hh p)*])

$$\begin{array}{l} pstate \ pstate \ ibT \ ibUT \\ (let \ l = (array-loc \ aa1 \ (nat \ (vs \ xx)) \ (Avstore \ as)) \\ in \ (Config \ 7 \ (State \ (Vstore \ (vs(vv := \ h \ l))) \ avst \ hh \ p))) \\ ibT \ ibUT \ [] \ ls \cup \ readLocs \ (Config \ pc \ (State \ (Vstore \ vs) \ avst \\ hh \ p)) \ ls] \end{array}$$

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

pstate pstate ibT ibUT
Config 8 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

pstate pstate ibT ibUT
*(let l = (array-loc aa2 (nat (vs vv * SS)) (Avstore as))*
in (Config 9 (State (Vstore (vs(tt := h l))) avst hh p)))
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore vs) avst
hh p)) ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

pstate pstate ibT ibUT
Config 10 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

pstate pstate ibT ibUT
Config 3 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*])

pstate pstate ibT ibUT
Config 12 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])

prefer 7 subgoal by metis by simp-all

by simp-all . . .

lemma *finalS-cond'*:*pcOf cfg < 12 ⇒ cfgs = [] ⇒ ibUT ≠ LNil ⇒ ¬ finalS*
(*pstate, cfg, cfgs, ibT, ibUT, ls*)

using *finalS-cond* **by** (*simp add: noMisSpec-def*)

lemma *finalS-while-spec*:

whileSpeculation cfg (last cfgs) ⇒
length cfgs = Suc 0 ⇒
¬ finalS (pstate, cfg, cfgs, ibT, ibUT, ls)
apply(*unfold whileSpec-defs, cases cfg*)

subgoal for $pc\ s$ **apply**($cases\ s$)
subgoal for $vst\ avst\ hh\ p$ **apply**($cases\ vst, cases\ avst, cases\ hh$)
subgoal for $vs\ as\ h$
apply($elim\ disjE, elim\ conjE$) **unfolding** $finalS-defs$
subgoal apply($cases\ resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$)
subgoal using $stepS-spec-resolve-iff$ [$of\ cfs\ pstate\ cfg\ ibT\ ibUT\ ls\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$] **by** $fastforce$
subgoal using $spec-resolve$ [$of\ cfs\ pstate\ cfg\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)\ cfg\ []\ ibT\ ibT\ ibUT\ ibUT\ ls\ ls$] **by** $fastforce$.
subgoal apply($cases\ resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$)
subgoal using $stepS-spec-resolve-iff$ [$of\ cfs\ pstate\ cfg\ ibT\ ibUT\ ls\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$] **by** $fastforce$
subgoal using $spec-resolve$ [$of\ cfs\ pstate\ cfg\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)\ cfg\ []\ ibT\ ibT\ ibUT\ ibUT\ ls\ ls$] **by** $fastforce\ \dots$.

lemma $finalS-while-spec-L2$:

$pcOf\ cfg = 6 \implies$
 $whileSpeculation\ (cfs!0)\ (last\ cfs) \implies$
 $length\ cfs = 2 \implies$
 $\neg\ finalS\ (pstate, cfg, cfs, ibT, ibUT, ls)$

apply($unfold\ whileSpec-defs, cases\ cfg$)

subgoal for $pc\ s$ **apply**($cases\ s$)
subgoal for $vst\ avst\ hh\ p$ **apply**($cases\ vst, cases\ avst, cases\ hh$)
subgoal for $vs\ as\ h$
apply($elim\ disjE, elim\ conjE$) **unfolding** $finalS-defs$
subgoal
apply($cases\ resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$)
subgoal using $stepS-spec-resolve-iff$ [$of\ cfs\ pstate\ cfg\ ibT\ ibUT\ ls\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$] **by** $fastforce$
subgoal using $spec-resolve$ [$of\ cfs\ pstate\ cfg\ -\ -\ -\ ibT\ -\ ibUT\ -\ -\ ls$] **by** $force$
.

subgoal
apply($cases\ resolve\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$)
subgoal using $stepS-spec-resolve-iff$ [$of\ cfs\ pstate\ cfg\ ibT\ ibUT\ ls\ update\ pstate\ (pcOf\ cfg\ \# \ map\ pcOf\ cfs)$] **by** $fastforce$
subgoal using $spec-resolve$ [$of\ cfs\ pstate\ cfg\ -\ -\ -\ ibT\ -\ ibUT\ -\ -\ ls$] **by** $force$
.

lemma $finalS-if-spec$:

$(pcOf\ (last\ cfs) \in inThenIfBeforeFence \wedge pcOf\ cfg = 10) \vee$
 $(pcOf\ (last\ cfs) \in inElseIf \wedge pcOf\ cfg = 6) \implies$
 $length\ cfs = Suc\ 0 \implies$
 $\neg\ finalS\ (pstate, cfg, cfs, ibT, ibUT, ls)$

unfolding $inThenIfBeforeFence-def\ inElseIf-def$

apply($simp, cases\ last\ cfs$)

subgoal for $pc\ s$ **apply**($cases\ s$)

subgoal for $vst\ avst\ hh\ p$ **apply**($cases\ vst, cases\ hh$)

subgoal for $vs\ h$

apply($elim\ disjE, elim\ conjE$) **unfolding** $finalS-defs$

```

subgoal apply(elim disjE)
  subgoal apply(cases resolve pstate (pcOf cfg # map pcOf cfigs))
    subgoal using spec-resolve[of cfigs pstate cfg - - - ibT - ibUT - - ls] using
cfigs-Suc-zero by blast
      subgoal apply(rule notI,
        erule allE[of - (pstate, cfig,
          [Config 7 (State (Vstore (vs(vv := h (array-loc aa1 (nat (vs
xx)) avst)))) avst hh p]],
          ibT, ibUT, ls ∪ readLocs (last cfigs)]])
        by(erule notE, rule spec-normal[of - - - - Config 7 (State (Vstore (vs(vv
:= h (array-loc aa1 (nat (vs xx)) avst)))) avst hh p]], auto) .
        by(metis cfigs-Suc-zero fence not-Cons-sel2 stepS-spec-Fence-iff spec-resolve)
      subgoal apply(elim conjE, elim disjE)
        subgoal apply(cases resolve pstate (pcOf cfg # map pcOf cfigs))
          subgoal using spec-resolve[of cfigs pstate cfg - - - ibT - ibUT - - ls] using
cfigs-Suc-zero by blast
            subgoal apply(rule notI,
              erule allE[of - (pstate, cfig,
                [Config 3 (State (Vstore vs) avst hh p]],
                ibT, ibUT, ls ∪ readLocs (last cfigs)]])
              by(erule notE, rule spec-normal[of - - - - Config 3 (State (Vstore vs)
avst hh p]], auto) .
            subgoal apply(cases resolve pstate (pcOf cfg # map pcOf cfigs))
              subgoal using spec-resolve[of cfigs pstate cfg - - - ibT - ibUT - - ls] using
cfigs-Suc-zero by blast
                subgoal apply(cases mispred pstate [6,3])
                  subgoal apply(rule notI, erule allE[of -
                    (update pstate (pcOf cfg # map pcOf cfigs),
                    cfig,
                    [Config (if vs xx ≠ 0 then 4 else 11) (State (Vstore vs) avst hh p),
                    Config (if vs xx ≠ 0 then 11 else 4) (State (Vstore vs) avst hh p)]],
                    ibT, ibUT,
                    ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p))]], erule notE,
                    rule spec-mispred[of - - - - -
                    (Config (if vs xx ≠ 0 then 4 else 11) (State (Vstore vs) avst hh p)
                    - - Config (if vs xx ≠ 0 then 11 else 4) (State (Vstore vs) avst hh p) ibT
                    ibUT)]
                    by(auto simp: finalM-iff)
                  apply(rule notI, erule allE[of -
                    (pstate, cfig, [Config (if vs xx ≠ 0 then 4 else 11) (State (Vstore vs) avst
                    hh p)], ibT, ibUT,
                    ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p))]])
                    by(erule notE, rule spec-normal[of - - - - Config (if vs xx ≠ 0 then 4
                    else 11) (State (Vstore vs) avst hh p)]], auto) .
                subgoal using spec-resolve[of cfigs pstate cfg - - - ibT - ibUT - - ls] by
fastforce
                subgoal using spec-resolve[of cfigs pstate cfg - - - ibT - ibUT - - ls] by

```

fastforce

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* = ∞ \wedge *llength* *ibUT2* = ∞ \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* *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*) \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 $\llbracket \rrbracket$ \wedge *ibUT2* \neq $\llbracket \rrbracket$ \wedge *w1 = w2*
unfolding *common-def PC-def* **by** (*auto simp: image-def subset-eq*)

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
 $\left. \right)$)

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 $\llbracket \rrbracket$ \wedge *ibUT2* \neq $\llbracket \rrbracket$ \wedge *cfgs3 = []* \wedge *cfgs4 = []*
apply (*meson* $\Delta 0$ -def *common-implies*)
by (*simp-all add: $\Delta 0$ -defs, metis Nil-is-map-conv*)

))
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, cfgs3, ibT3, ibUT3, ls3)
 (pstate4, cfg4, cfgs4, 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 cfgs3) \wedge
 whileSpeculation cfg4 (last cfgs4) \wedge
 length cfgs3 = Suc 0 \wedge length cfgs4 = Suc 0
unfolding $\Delta 1'$ -defs
apply (simp add: lessI, clarify)
by (metis last-map length-0-conv)

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)
 (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 = startOfIfThen \wedge pcOf (last cfgs3) \in inElseIf \wedge
 misSpecL1 cfgs3 \wedge misSpecL1 cfgs4 \wedge

 (pcOf (last cfgs3) = startOfElseBranch \longrightarrow w1 = ∞) \wedge
 (pcOf (last cfgs3) = 3 \longrightarrow w1 = 3) \wedge

 (pcOf (last cfgs3) = startOfWhileThen \vee
 pcOf (last cfgs3) = 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 (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 inElseIf \wedge pcOf cfg3 = 6 \wedge
 pcOf (last cfgs4) = pcOf (last cfgs3) \wedge
 pcOf cfg4 = pcOf cfg3 \wedge length cfgs3 = Suc 0 \wedge
 length cfgs4 = Suc 0 \wedge same-var xx (last cfgs3) (last cfgs4)
apply(intro conjI)
unfolding $\Delta 2$ -defs
apply (simp-all add: image-subset-iff)
by (metis last-in-set length-0-conv Nil-is-map-conv last-map length-map)+

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)
 (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 = 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

$pcOf\ cfg3 = 6 \wedge pcOf\ cfg4 = 6 \wedge$
 $pcOf\ (last\ cfigs3) = pcOf\ (last\ cfigs4) \wedge$
 $whileSpeculation\ (cfigs3!0)\ (last\ cfigs3) \wedge$
 $whileSpeculation\ (cfigs4!0)\ (last\ cfigs4) \wedge$
 $length\ cfigs3 = 2 \wedge length\ cfigs4 = 2$
apply(*intro conjI*)
unfolding $\Delta 2'$ -*defs* **apply** (*simp add: lessI, clarify*)
apply *linarith+* **apply** *simp-all*
apply (*metis last-map length-0-conv*)
by (*metis list.inject map-L2*)

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

$\Delta 3 = (\lambda num\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$

$(common\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$

$same-var-o\ xx\ cfig3\ cfigs3\ cfg4\ cfigs4 \wedge$
 $pcOf\ cfig3 = startOfElseBranch \wedge pcOf\ (last\ cfigs3) \in inThenIfBeforeFence \wedge$
 $misSpecL1\ cfigs3 \wedge$
 $(pcOf\ (last\ cfigs3) = 6 \longrightarrow w1 = \infty) \wedge$
 $(pcOf\ (last\ cfigs3) = 7 \longrightarrow w1 = 1)$

))

lemmas $\Delta 3$ -*defs* = $\Delta 3$ -*def* *common-def* *PC-def* *same-var-o-def*
startOfElseBranch-def *inThenIfBeforeFence-def*

lemma $\Delta 3$ -*implies*: $\Delta 3\ num\ w1\ w2\ (pstate3, cfig3, cfigs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfigs4, ibT4, ibUT4, ls4)$
 $statA$

$(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$

$statO \implies$

$pcOf\ (last\ cfigs3) \in inThenIfBeforeFence \wedge$
 $pcOf\ (last\ cfigs4) = pcOf\ (last\ cfigs3) \wedge$
 $pcOf\ cfig3 = 10 \wedge pcOf\ cfig3 = pcOf\ cfg4 \wedge$
 $length\ cfigs3 = Suc\ 0 \wedge length\ cfigs4 = Suc\ 0$

apply(*intro conjI*)

unfolding $\Delta 3$ -*defs* *misSpecL1-def*

apply (*simp-all add: image-subset-iff*)

by (metis last-map map-is-Nil-conv length-map)+

definition $\Delta e :: \text{enat} \Rightarrow \text{enat} \Rightarrow \text{enat} \Rightarrow \text{stateO} \Rightarrow \text{stateO} \Rightarrow \text{status} \Rightarrow \text{stateV} \Rightarrow \text{stateV} \Rightarrow \text{status} \Rightarrow \text{bool}$ **where**
 $\Delta e = (\lambda \text{num } w1 \ w2 \ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $\quad (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $\quad \text{statA}$
 $\quad (cfg1, ibT1, ibUT1, ls1)$
 $\quad (cfg2, ibT2, ibUT2, ls2)$
 $\quad \text{statO}.$
 $(pcOf \ cfg3 = \text{endPC} \wedge pcOf \ cfg4 = \text{endPC} \wedge cfs3 = [] \wedge cfs4 = [] \wedge$
 $\quad pcOf \ cfg1 = \text{endPC} \wedge pcOf \ cfg2 = \text{endPC}))$

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

lemma *init: initCond* $\Delta 0$

unfolding *initCond-def* **apply** *safe*

subgoal for $pstate3 \ cfg3 \ cfs3 \ ibT3 \ ibUT3 \ ls3 \ pstate4 \ cfg4 \ cfs4 \ ibT4 \ ibUT4 \ ls4$

unfolding *istateO.simps* **apply** *clarsimp*

apply(cases *getAvstore* (*stateOf* $cfg3$), cases *getAvstore* (*stateOf* $cfg4$))

unfolding $\Delta 0\text{-defs}$

unfolding *array-base-def* **by** *auto* .

lemma *step0: unwindIntoCond* $\Delta 0$ (*oor* $\Delta 0 \ \Delta 1$)

proof(rule *unwindIntoCond-simpleI*)

fix $n \ w1 \ w2 \ ss3 \ ss4 \ \text{statA} \ ss1 \ ss2 \ \text{statO}$

assume $r: \text{reachO} \ ss3 \ \text{reachO} \ ss4 \ \text{reachV} \ ss1 \ \text{reachV} \ ss2$

and $\Delta 0: \Delta 0 \ n \ w1 \ w2 \ ss3 \ ss4 \ \text{statA} \ ss1 \ ss2 \ \text{statO}$

obtain $pstate3 \ cfg3 \ cfs3 \ ibT3 \ ibUT3 \ ls3$ **where** $ss3: ss3 = (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

by (cases $ss3$, *auto*)

obtain $pstate4 \ cfg4 \ cfs4 \ ibT4 \ ibUT4 \ ls4$ **where** $ss4: ss4 = (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$

by (cases $ss4$, *auto*)

obtain $cfg1 \ ibT1 \ ibUT1 \ ls1$ **where** $ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (cases $ss1$, *auto*)

obtain $cfg2 \ ibT2 \ ibUT2 \ ls2$ **where** $ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (cases $ss2$, *auto*)

note $ss = ss3 \ ss4 \ ss1 \ ss2$

```

obtain  $pc3\ vs3\ avst3\ h3\ p3$  where
 $cfg3: cfg3 = Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)$ 
by ( $cases\ cfg3$ ) ( $metis\ state.collapse\ vstore.collapse$ )
obtain  $pc4\ vs4\ avst4\ h4\ p4$  where
 $cfg4: cfg4 = Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)$ 
by ( $cases\ cfg4$ ) ( $metis\ state.collapse\ vstore.collapse$ )
note  $cfg = cfg3\ cfg4$ 

obtain  $hh3$  where  $h3: h3 = Heap\ hh3$  by( $cases\ h3, auto$ )
obtain  $hh4$  where  $h4: h4 = Heap\ hh4$  by( $cases\ h4, auto$ )
note  $hh = h3\ h4$ 

have  $f1: \neg finalN\ ss1$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-by( $frule\ \Delta 0\ implies, simp$ )

have  $f2: \neg finalN\ ss2$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-by( $frule\ \Delta 0\ implies, simp$ )

have  $f3: \neg finalS\ ss3$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-apply( $frule\ \Delta 0\ implies, unfold\ \Delta 0\ defs$ )
  by ( $clarify, metis\ finalS\ cond'$ )

have  $f4: \neg finalS\ ss4$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-apply( $frule\ \Delta 0\ implies, unfold\ \Delta 0\ defs$ )
  by ( $clarify, metis\ finalS\ cond'$ )

note  $finals = f1\ f2\ f3\ f4$ 
show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
  using  $finals$  by  $auto$ 

then show  $isIntO\ ss3 = isIntO\ ss4$  by  $simp$ 

show  $react\ (oor\ \Delta 0\ \Delta 1)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $react\ def$  proof( $intro\ conjI$ )

  show  $match1\ (oor\ \Delta 0\ \Delta 1)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match1\ def$  by ( $simp\ add: finalS\ def\ final\ def$ )
  show  $match2\ (oor\ \Delta 0\ \Delta 1)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match2\ def$  by ( $simp\ add: finalS\ def\ final\ def$ )
  show  $match12\ (oor\ \Delta 0\ \Delta 1)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 

proof( $rule\ match12\ simpleI, rule\ disjI2, intro\ conjI$ )
  fix  $ss3'\ ss4'\ statA'$ 
  assume  $statA': statA' = sstatA'\ statA\ ss3\ ss4$ 

```

```

and  $v$ : validTransO ( $ss3$ ,  $ss3'$ ) validTransO ( $ss4$ ,  $ss4'$ )
and  $sa$ : Opt.eqAct  $ss3$   $ss4$ 
note  $v3 = v(1)$  note  $v4 = v(2)$ 

obtain  $pstate3'$   $cfg3'$   $cfgs3'$   $ibT3'$   $ibUT3'$   $ls3'$  where  $ss3'$ :  $ss3' = (pstate3'$ ,
 $cfg3'$ ,  $cfgs3'$ ,  $ibT3'$ ,  $ibUT3'$ ,  $ls3')$ 
by (cases  $ss3'$ , auto)
obtain  $pstate4'$   $cfg4'$   $cfgs4'$   $ibT4'$   $ibUT4'$   $ls4'$  where  $ss4'$ :  $ss4' = (pstate4'$ ,
 $cfg4'$ ,  $cfgs4'$ ,  $ibT4'$ ,  $ibUT4'$ ,  $ls4')$ 
by (cases  $ss4'$ , auto)
note  $ss = ss\ ss3'\ ss4'$ 

obtain  $pc3$   $vs3$   $avst3$   $h3$   $p3$  where
 $cfg3$ :  $cfg3 = \text{Config } pc3\ (\text{State } (Vstore\ vs3))\ avst3\ h3\ p3)$ 
by (cases  $cfg3$ ) (metis state.collapse vstore.collapse)
obtain  $pc4$   $vs4$   $avst4$   $h4$   $p4$  where
 $cfg4$ :  $cfg4 = \text{Config } pc4\ (\text{State } (Vstore\ vs4))\ avst4\ h4\ p4)$ 
by (cases  $cfg4$ ) (metis state.collapse vstore.collapse)
note  $cfg = cfg3\ cfg4$ 

show eqSec  $ss1$   $ss3$ 
using  $v$   $sa$   $\Delta 0$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)

show eqSec  $ss2$   $ss4$ 
using  $v$   $sa$   $\Delta 0$  unfolding  $ss$ 
apply (simp add:  $\Delta 0$ -defs)
by (metis map-is-Nil-conv)

show Van.eqAct  $ss1$   $ss2$ 
using  $v$   $sa$   $\Delta 0$  unfolding  $ss$ 
apply-apply(frule  $\Delta 0$ -implies)
unfolding Opt.eqAct-def
Van.eqAct-def
by(simp-all add:  $\Delta 0$ -defs, linarith)

show match12-12 (oor  $\Delta 0$   $\Delta 1$ )  $\infty \infty$   $ss3'$   $ss4'$   $statA'$   $ss1$   $ss2$  statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
show validTransV ( $ss1$ , nextN  $ss1$ )
by (simp add: f1 nextN-stepN)

show validTransV ( $ss2$ , nextN  $ss2$ )
by (simp add: f2 nextN-stepN)

{assume  $sstat$ :  $statA' = \text{Diff}$ 
show  $sstatO'$   $statO$   $ss1$   $ss2 = \text{Diff}$ 
using  $v$   $sa$   $\Delta 0$  sstat unfolding  $ss$   $cfg$   $statA'$  apply simp
apply(simp add:  $\Delta 0$ -defs sstatO'-def sstatA'-def finalS-def final-def)

```

```

using cases-12[of pc3] apply(elim disjE)
apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
apply(cases statO, simp-all) apply (cases statA, simp-all)
by (smt (z3) status.distinct(1) newStat.simps(2,3) newStat-diff)+
} note stat = this

have cfgs:cfgs3 = [] cfgs4 = [] using  $\Delta 0$  unfolding ss apply-by(frule
 $\Delta 0$ -implies, auto)+

show oor  $\Delta 0$   $\Delta 1$   $\infty$   $\infty$   $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case nonspec-mispred
then show ?thesis using sa  $\Delta 0$  stat unfolding ss
by (simp add:  $\Delta 0$ -defs numeral-1-eq-Suc-0, linarith)
next
case nonspec-normal note nn3 = nonspec-normal
show ?thesis
using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case nonspec-mispred
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-normal
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-mispred
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-Fence
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case spec-resolve
then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
next
case nonspec-normal note nn4 = nonspec-normal
show ?thesis using sa  $\Delta 0$  stat v3 v4 nn3 nn4 unfolding ss cfg apply
clarsimp
apply(unfold  $\Delta 0$ -defs, clarsimp, elim disjE)
subgoal by(rule oorI1, auto simp add:  $\Delta 0$ -defs)
subgoal by (rule oorI1, simp add:  $\Delta 0$ -defs)
subgoal by (rule oorI2, simp add:  $\Delta 1$ -defs) .
qed(simp-all add: cfgs)
qed(simp-all add: cfgs)

```

qed
 qed
 qed
 qed

lemma *step1: unwindIntoCond* $\Delta 1$ (*oor5* $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe)

proof(*rule unwindIntoCond-simpleI*)

fix $n w1 w2 ss3 ss4 statA ss1 ss2 statO$

assume $r: reachO ss3 reachO ss4 reachV ss1 reachV ss2$

and $\Delta 1: \Delta 1 n w1 w2 ss3 ss4 statA ss1 ss2 statO$

obtain $pstate3 cfg3 cfs3 ibT3 ibUT3 ls3$ **where** $ss3: ss3 = (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

by (*cases ss3, auto*)

obtain $pstate4 cfg4 cfs4 ibT4 ibUT4 ls4$ **where** $ss4: ss4 = (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$

by (*cases ss4, auto*)

obtain $cfg1 ibT1 ibUT1 ls1$ **where** $ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (*cases ss1, auto*)

obtain $cfg2 ibT2 ibUT2 ls2$ **where** $ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (*cases ss2, auto*)

note $ss = ss3 ss4 ss1 ss2$

obtain $pc3 vs3 avst3 h3 p3$ **where**

$cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)$

by (*cases cfg3*) (*metis state.collapse vstore.collapse*)

obtain $pc4 vs4 avst4 h4 p4$ **where**

$cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)$

by (*cases cfg4*) (*metis state.collapse vstore.collapse*)

note $cfg = cfg3 cfg4$

obtain $hh3$ **where** $h3: h3 = Heap hh3$ **by**(*cases h3, auto*)

obtain $hh4$ **where** $h4: h4 = Heap hh4$ **by**(*cases h4, auto*)

note $hh = h3 h4$

have $f1: \neg finalN ss1$

using $\Delta 1$ **unfolding** $ss \Delta 1-def$ **apply** *clarify*

apply(*frule common-implies*)

using *finalB-pcOf-iff finalN-iff-finalB nat-less-le* **by** *blast*

have $f2: \neg finalN ss2$

using $\Delta 1$ **unfolding** $ss \Delta 1-def$ **apply** *clarify*

apply(*frule common-implies*)

using *finalB-pcOf-iff finalN-iff-finalB nat-less-le* **by** *metis*

have $f3: \neg finalS ss3$

```

using  $\Delta 1$  unfolding ss
apply-apply(frul  $\Delta 1$ -implies)
by (simp add: finalS-cond')

have f4:¬finalS ss4
  using  $\Delta 1$  unfolding ss
  apply-apply(frul  $\Delta 1$ -implies)
  by (simp add: finalS-cond')

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

have cfs: cfs3 = [] cfs4 = [] using  $\Delta 1$  unfolding ss apply-by(frul  $\Delta 1$ -implies,
auto)+

show react (oor5  $\Delta 1$   $\Delta 1'$   $\Delta 2$   $\Delta 3$   $\Delta e$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 (oor5  $\Delta 1$   $\Delta 1'$   $\Delta 2$   $\Delta 3$   $\Delta e$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor5  $\Delta 1$   $\Delta 1'$   $\Delta 2$   $\Delta 3$   $\Delta e$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 (oor5  $\Delta 1$   $\Delta 1'$   $\Delta 2$   $\Delta 3$   $\Delta e$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO

  proof(rule match12-simpleI, rule disjI2, intro conjI)
    fix ss3' ss4' statA'
    assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfs3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfs3', cfs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfs4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfs4', cfs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

  show eqSec ss1 ss3
  using v sa  $\Delta 1$  unfolding ss by (simp add:  $\Delta 1$ -defs)

  show eqSec ss2 ss4
  using v sa  $\Delta 1$  unfolding ss by (simp add:  $\Delta 1$ -defs)

```

```

show Van.eqAct ss1 ss2
using v sa Δ1 unfolding ss
unfolding Opt.eqAct-def Van.eqAct-def
apply(simp-all add: Δ1-defs)
by (metis Nil-is-map-conv f3 infinity-ne-i0 llength-LNil ss3)

show match12-12 (oor5 Δ1 Δ1' Δ2 Δ3 Δe) ∞ ∞ ss3' ss4' statA' ss1 ss2
statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
    using v sa Δ1 sstat finals unfolding ss cfg statA'
    apply–apply(frule Δ1-implies)
    apply(simp add: Δ1-defs sstatO'-def sstatA'-def newStat-EqI)
    using cases-12[of pc3] apply(elim disjE, simp-all)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    subgoal apply(cases statO, simp-all, cases statA)
      by (simp-all add: newStat-EqI split: if-splits)
    subgoal apply(cases statO, simp-all)
      by(cases statA, simp-all add: newStat-EqI)
    apply(cases statO, simp-all, cases statA)
      by (simp-all add: newStat-EqI split: if-splits)
  } note stat = this

  show oor5 Δ1 Δ1' Δ2 Δ3 Δe ∞ ∞ ∞ ss3' ss4' statA' (nextN ss1)
(nextN ss2) (sstatO' statO ss1 ss2)

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
case spec-normal

```

```

then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-mispred
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-Fence
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-resolve
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4 [unfolded ss, simplified] proof (cases rule: stepS-cases)

    case nonspec-mispred
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-normal
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-mispred
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-resolve
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case nonspec-normal note nn4 = nonspec-normal
      then show ?thesis using sa  $\Delta 1$  stat v3 v4 nn3 nn4 f4 unfolding ss cfg
Opt.eqAct-def
      apply clarsimp using cases-12 [of pc3] apply (elim disjE)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal using xx-0-cases [of vs3] apply (elim disjE)
      subgoal by (rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      subgoal by (rule oor5I1, auto simp add:  $\Delta 1$ -defs) .
      subgoal apply (rule oor5I1) by (auto simp add:  $\Delta 1$ -defs)

```

```

    subgoal using xx-NN-cases[of vs3] apply(elim disjE)
      subgoal by(rule oor5I1, auto simp add: Δ1-defs)
      subgoal by(rule oor5I1, auto simp add: Δ1-defs) .
    subgoal by(rule oor5I1, auto simp add: Δ1-defs hh)
    subgoal by(rule oor5I1, auto simp add: Δ1-defs)
    subgoal by(rule oor5I1, auto simp add: Δ1-defs hh)
    subgoal by(rule oor5I1, auto simp add: Δ1-defs)
    subgoal by(rule oor5I1, auto simp add: Δ1-defs)
    by(rule oor5I5, simp-all add: Δ1-defs Δe-defs)
  qed(simp-all add: cfs)
next
  case nonspec-mispred note nm3 = nonspec-mispred
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

    case nonspec-normal
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-normal
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-mispred
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-Fence
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case spec-resolve
    then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
    case nonspec-mispred note nm4 = nonspec-mispred
    then show ?thesis using sa Δ1 stat v3 v4 nm3 nm4 unfolding ss cfg
apply clarsimp
    using cases-12[of pc3] apply(elim disjE)
      prefer 4 subgoal using xx-0-cases[of vs3] apply(elim disjE)
        subgoal by(rule oor5I2, auto simp add: Δ1-defs Δ1'-defs)
        subgoal by(rule oor5I2, auto simp add: Δ1-defs Δ1'-defs) .
      prefer 5 subgoal using xx-NN-cases[of vs3] apply(elim disjE)
        subgoal apply(rule oor5I3) by (auto simp add: Δ1-defs Δ2-defs)
        subgoal apply(rule oor5I4) by (auto simp add: Δ1-defs Δ3-defs) .
      by (simp-all add: Δ1-defs)
    qed(simp-all add: cfs)
  qed(simp-all add: cfs)
qed
qed

```

qed
qed

lemma *step2: unwindIntoCond* $\Delta 2$ (*oor3* $\Delta 2$ $\Delta 2'$ $\Delta 1$)

proof(*rule unwindIntoCond-simpleI*)

fix $n w1 w2 ss3 ss4 statA ss1 ss2 statO$

assume $r: reachO ss3 reachO ss4 reachV ss1 reachV ss2$

and $\Delta 2: \Delta 2 n w1 w2 ss3 ss4 statA ss1 ss2 statO$

obtain $pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3$ **where** $ss3: ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by (*cases ss3, auto*)

obtain $pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4$ **where** $ss4: ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by (*cases ss4, auto*)

obtain $cfg1 ibT1 ibUT1 ls1$ **where** $ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (*cases ss1, auto*)

obtain $cfg2 ibT2 ibUT2 ls2$ **where** $ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (*cases ss2, auto*)

note $ss = ss3 ss4 ss1 ss2$

obtain $pc3 vs3 avst3 h3 p3$ **where**

$lcfgs3: last cfgs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)$

by (*cases last cfgs3*) (*metis state.collapse vstore.collapse*)

obtain $pc4 vs4 avst4 h4 p4$ **where**

$lcfgs4: last cfgs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)$

by (*cases last cfgs4*) (*metis state.collapse vstore.collapse*)

note $lcfgs = lcfgs3 lcfgs4$

have $f1: \neg finalN ss1$

using $\Delta 2$ **unfolding** $ss \Delta 2-def$

apply *clarsimp*

by(*frule common-implies, simp*)

have $f2: \neg finalN ss2$

using $\Delta 2$ **unfolding** $ss \Delta 2-def$

apply *clarsimp*

by(*frule common-implies, simp*)

have $f3: \neg finalS ss3$

using $\Delta 2$ **unfolding** ss

apply—**apply**(*frule* $\Delta 2-implies$)

by (*simp add: finalS-if-spec*)

have $f4: \neg finalS ss4$

using $\Delta 2$ **unfolding** ss

apply-apply(*frule* $\Delta 2$ -*implies*)
by (*simp add: finalS-if-spec*)

note *finals* = *f1 f2 f3 f4*
show *finalS ss3* = *finalS ss4* \wedge *finalN ss1* = *finalS ss3* \wedge *finalN ss2* = *finalS ss4*
using *finals* **by** *auto*

then show *isIntO ss3* = *isIntO ss4* **by** *simp*

then have *lpc3:pcOf (last cfgs3) = 10* \vee
pcOf (last cfgs3) = 3 \vee
pcOf (last cfgs3) = 4 \vee
pcOf (last cfgs3) = 11
using $\Delta 2$ **unfolding** *ss* $\Delta 2$ -*defs* **by** *simp*

have *sec3[simp]: \neg isSecO ss3*
using $\Delta 2$ **unfolding** *ss* **by** (*simp add: $\Delta 2$ -defs*)
have *sec4[simp]: \neg isSecO ss4*
using $\Delta 2$ **unfolding** *ss* **by** (*simp add: $\Delta 2$ -defs*)

have *stat[simp]: \wedge ss3' s4' statA'. statA' = sstatA' statA ss3 ss4* \implies
validTransO (ss3, s3') \implies *validTransO (ss4, s4')* \implies
(statA = statA' \vee statO = Diff)

subgoal for *ss3' ss4'*
apply (*cases ss3, cases ss4, cases ss1, cases ss2*)
apply (*cases ss3', cases ss4', clarsimp*)
using $\Delta 2$ *finals unfolding ss* **apply** *clarsimp*
apply(*simp-all add: $\Delta 2$ -defs sstatA'-def*)
apply(*cases statO, simp-all*) **by** (*cases statA, simp-all add: newStat-EqI*) .

have *xx:vs3 xx = vs4 xx* **using** $\Delta 2$ *lcfgs unfolding ss* $\Delta 2$ -*defs* **apply** *clarsimp*
by (*metis cfgs-Suc-zero config.sel(2) list.set-intros(1) state.sel(1) vstore.sel*)

have *oor3-rule: \wedge ss3' ss4'. ss3 \rightarrow_S ss3' \implies ss4 \rightarrow_S ss4' \implies*
(pcOf (last cfgs3) = 10 \implies oor3 $\Delta 2$ $\Delta 2'$ $\Delta 1 \infty 3 3$ ss3' ss4'
(sstatA' statA ss3 ss4) ss1 ss2 statO)
 \wedge (*pcOf (last cfgs3) = 3 \wedge mispred pstate4 [6, 3] \implies oor3 $\Delta 2$ $\Delta 2'$*
 $\Delta 1 \infty 2 2$ ss3' ss4' (sstatA' statA ss3 ss4) ss1 ss2 statO)
 \wedge (*pcOf (last cfgs3) = 3 \wedge \neg mispred pstate4 [6, 3] \implies oor3 $\Delta 2$*
 $\Delta 2' \Delta 1 \infty 1 1$ ss3' ss4' (sstatA' statA ss3 ss4) ss1 ss2 statO)
 \wedge (*(pcOf (last cfgs3) = 4 \vee pcOf (last cfgs3) = 11) \implies oor3 $\Delta 2$*
 $\Delta 2' \Delta 1 \infty 0 0$ ss3' ss4' (sstatA' statA ss3 ss4) ss1 ss2 statO) \implies
 $\exists w1' < w1. \exists w2' < w2. \text{oor3 } \Delta 2 \Delta 2' \Delta 1 \infty w1' w2' \text{ ss3' ss4'}$
(sstatA' statA ss3 ss4) ss1 ss2 statO
subgoal for *ss3' ss4'* **apply**(*cases ss3', cases ss4'*)
subgoal for *pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3'*
pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4'
subgoal premises *p* **using** *lpc3* **apply-apply**(*erule disjE*)

```

subgoal apply(intro exI[of - 3], intro conjI)
  subgoal using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
    by (metis enat-ord-simps(4) numeral-ne-infinity)
  apply(intro exI[of - 3], rule conjI)
  subgoal using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
    by (metis enat-ord-simps(4) numeral-ne-infinity)
  using p by (simp add: p)
apply(erule disjE)
subgoal apply(cases mispred pstate4 [6, 3])
  subgoal apply(intro exI[of - 2], intro conjI)
    using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
      apply (metis enat-ord-number(2) eval-nat-numeral(3) lessI)
    apply(intro exI[of - 2], rule conjI)
    using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
      apply (metis enat-ord-number(2) eval-nat-numeral(3) lessI)
    using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify
  subgoal apply(intro exI[of - 1], intro conjI)
using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis one-less-numeral-iff semiring-norm(77))
apply(intro exI[of - 1], rule conjI)
using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis one-less-numeral-iff semiring-norm(77))
using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify .
subgoal apply(intro exI[of - 0], intro conjI)
  using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
    apply (metis less-numeral-extra(1))
  apply(intro exI[of - 0], rule conjI)
using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis less-numeral-extra(1))
using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify . . . .

have pstate3:pstate3 = pstate4 using  $\Delta 2$ [unfolded ss  $\Delta 2$ -defs] by fast
have pcc:pcOf (last cfs3) = pcOf (last cfs4) using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded
ss]] by auto
show react (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

show match1 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-simpleI, simp-all, rule disjI1)
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
  subgoal for pstate3' cfs3' cfs3' ibT3' ibUT3' ls3'
    pstate4' cfs4' cfs4' ibT4' ibUT4' ls4'
  apply-apply(rule oor3-rule, assumption+, intro conjI impI)

subgoal premises prem using prem(1)[unfolded ss prem(4)]

```

```

proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case spec-mispred
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-Fence
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-resolve note sr3 = spec-resolve
  have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sr3  $\Delta 2$ -implies[OF
 $\Delta 2$ [unfolded ss]] pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case spec-Fence then show ?thesis using prem(6) pcc by auto
  next
    case spec-mispred then show ?thesis using prem(6) pcc by simp
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolve note sr4 = spec-resolve
    have pc4:pc4 = 10 using  $\Delta 2$  prem lcfgs unfolding ss  $\Delta 2$ -defs by auto
    show ?thesis
    using stat  $\Delta 2$  sr3 sr4 prem
    unfolding ss lcfgs
    apply-apply(rule oor3I3)
    apply(frule  $\Delta 2$ -implies, simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
    by clarsimp
  qed(simp add: r4)
next
  case spec-normal note sn3 = spec-normal
  have r4: $\neg$  resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sn3

```

```

Δ2-implies[OF Δ2[unfolded ss]] pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sn3 Δ2 unfolding ss by (simp add: Δ2-defs)
  next
    case nonspec-mispred
    then show ?thesis using sn3 Δ2 unfolding ss by (simp add: Δ2-defs)
  next
    case spec-Fence
    then show ?thesis using sn3 Δ2 unfolding ss by (simp add: Δ2-defs,
metis last-map)
  next
    case spec-resolve
    then show ?thesis using r4 by auto
  next
    case spec-mispred
    then show ?thesis using sn3 Δ2 unfolding ss by (simp add: Δ2-defs,
metis last-map)
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-normal note sn4 = spec-normal
    have pc4:pc4 = 10 using Δ2 prem lcfgs unfolding ss Δ2-defs by auto
    show ?thesis
      using Δ2 prem sn3 sn4 finals stat unfolding ss prem(4,5) lcfgs
      apply-apply(frul Δ2-implies, unfold Δ2-defs) apply clarsimp
      apply(rule oor3I1) apply(simp-all add: Δ2-defs pc4)
      using final-def config.sel(2) last-in-set
      lcfgs state.sel(1,2) vstore.sel xx
      by (metis (mono-tags, lifting))
    qed
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  qed

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat Δ2 prem unfolding ss by (auto simp add:
Δ2-defs)

```

```

next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case spec-Fence
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-normal
  have mispred pstate3 (pcOf cfg3 # map pcOf cfgs3) using  $\Delta 2$  prem
unfolding ss pstate3 by (auto simp add:  $\Delta 2$ -defs)
  then show ?thesis using prem(6) spec-normal(3) unfolding pstate3 by
(auto)
next
  case spec-resolve note sr3 = spec-resolve
  have r4: resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sr3  $\Delta 2$ -implies[OF
 $\Delta 2$ [unfolded ss]] pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case spec-Fence then show ?thesis using prem(6) pcc by auto
  next
    case spec-mispred then show ?thesis using r4 by simp
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolve note sr4 = spec-resolve
    have pc4: pc4 = 3 using  $\Delta 2$  prem lcfgs unfolding ss  $\Delta 2$ -defs by auto
    show ?thesis
    using stat  $\Delta 2$  sr3 sr4 prem
    unfolding ss lcfgs
    apply-apply(rule oor3I3)
    apply(frule  $\Delta 2$ -implies, simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
    by clarsimp
    qed(simp add: r4)
  next
    case spec-mispred note sm3 = spec-mispred
    have r4:  $\neg$  resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sm3
 $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] pstate3 by auto
    show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)

```

```

    case nonspec-normal
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case nonspec-mispred
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolve
  then show ?thesis using r4 by auto
next
  case spec-Fence
  then show ?thesis using sm3  $\Delta 2$  unfolding ss apply-apply(frule
 $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs)
next
  case spec-normal
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
  case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
next
  case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
next
  case spec-mispred note sm4 = spec-mispred
  have pc:pc4 = 3
  using prem(6) lcfs  $\Delta 2$  unfolding ss apply-apply(frule  $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs )
  show ?thesis apply(rule oor3I2)
  unfolding ss  $\Delta 2'$ -def using xx-0-cases[of vs3] apply(elim disjE)
  subgoal using  $\Delta 2$  lcfs prem pc sm3 sm4 xx finals stat unfolding ss
  apply- apply(simp add:  $\Delta 2$ -defs  $\Delta 2'$ -defs, clarify)
  apply(intro conjI)
  subgoal by (metis config.sel(2) last-in-set state.sel(1,2) vstore.sel
final-def)
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (smt (verit) prem(1) prem(2) ss3 ss4)
  subgoal by (metis config.sel(2) last-in-set state.sel(1) vstore.sel) .
  subgoal using  $\Delta 2$  lcfs prem pc sm3 sm4 xx finals stat unfolding ss
  apply- apply(simp add:  $\Delta 2$ -defs  $\Delta 2'$ -defs, clarify)
  apply(intro conjI)
  subgoal by (metis config.sel(2) last-in-set state.sel(1,2) vstore.sel
final-def)
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (smt (verit) prem(1) prem(2) ss3 ss4)

```

```

      subgoal by (metis config.sel(2) last-in-set state.sel(1) vstore.sel) . .
    qed
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  qed

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
    case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

  next
    case spec-Fence
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
  next
    case spec-mispred
  have  $\neg$ mispred pstate3 (pcOf cfg3 # map pcOf cfgs3) using  $\Delta 2$ -implies[OF
 $\Delta 2$ [unfolded ss]] pstate3 by (simp add: prem(6))
  then show ?thesis using spec-mispred by auto
  next
    case spec-resolve note sr3 = spec-resolve
  have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sr3  $\Delta 2$ -implies[OF
 $\Delta 2$ [unfolded ss]] pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sr3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case spec-Fence then show ?thesis using prem(6) pcc by auto
  next
    case spec-mispred then show ?thesis using r4 by simp
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next

```

```

case spec-resolve note sr4 = spec-resolve
have pc4:pc4 = 3 using  $\Delta 2$  prem lcfgs unfolding ss  $\Delta 2$ -defs by auto
show ?thesis
using stat  $\Delta 2$  sr3 sr4 prem
unfolding ss lcfgs
apply-apply(rule oor3I3)
apply(frule  $\Delta 2$ -implies, simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
by clarsimp
qed(simp add: r4)
next
case spec-normal note sn3 = spec-normal
have r4: $\neg$  resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sn3
 $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] pstate3 by auto
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
case nonspec-normal
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
case nonspec-mispred
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
case spec-Fence
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
case spec-resolve
then show ?thesis using r4 by auto
next
case spec-mispred
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
case spec-normal note sn4 = spec-normal
show ?thesis
using  $\Delta 2$  lcfgs prem sn3 sn4 finals unfolding ss
apply-apply(frule  $\Delta 2$ -implies) apply clarify
apply(rule oor3I1, clarsimp)
using xx-0-cases[of vs3] apply(elim disjE)
subgoal apply(simp-all add:  $\Delta 2$ -defs)
using config.sel(2) last-in-set stat state.sel(1,2) vstore.sel
by (smt (verit, ccfv-SIG) Opt.final-def config.sel(1) eval-nat-numeral(3)
f3 f4 is-Output-1 le-imp-less-Suc le-refl nat-less-le ss)
subgoal apply(simp-all add:  $\Delta 2$ -defs, clarify)
using config.sel(2) last-in-set stat state.sel(1,2) vstore.sel
apply(intro conjI, unfold config.sel(1))
subgoal by simp
subgoal by simp
subgoal by (metis array-baseSimp)
subgoal by (metis array-baseSimp)
subgoal by (metis array-baseSimp)

```

```

      subgoal by (metis array-baseSimp)
      subgoal by (smt (verit) cfgs-Suc-zero lcfgs list.set-intros(1))
      subgoal by (smt (verit) cfgs-Suc-zero lcfgs list.set-intros(1))
      subgoal by (smt (z3) Opt.final-def ss3 ss4)
      subgoal by (smt (z3) cfgs-Suc-zero lcfgs3 list.set-intros(1))
      subgoal by (smt (z3) cfgs-Suc-zero lcfgs3 list.set-intros(1))
      subgoal by linarith
      subgoal by linarith
      subgoal by linarith . .
    next
      case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
    next
      case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
    qed
  next
    case spec-resolveO then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  next
    case spec-resolveI then show ?thesis using prem(6) pcc is-getInput-pcOf
is-Output-pcOf by auto
  qed

  subgoal premises prem using prem(1)[unfolded ss prem(4)]
  proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

  next
    case spec-Fence
    then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
    case spec-mispred
    then show ?thesis using  $\Delta 2$  prem unfolding ss by auto
  next
    case spec-normal
    then show ?thesis using  $\Delta 2$  prem unfolding ss by auto
  next
    case spec-resolve note sr3 = spec-resolve
    then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) cfgs4  $\neq$  []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
    show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
      case nonspec-normal

```

```

then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case nonspec-mispred
then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case spec-normal
  then show ?thesis using r4 by auto
next
  case spec-mispred
  then show ?thesis using r4 by auto
next
  case spec-Fence
  then show ?thesis using r4 by auto
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply-
  apply(frul  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (smt(verit) prem(1) prem(2) ss)
next
  case spec-resolveO note sr4 = spec-resolveO
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply-
  apply(frul  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (smt(verit) prem(1) prem(2) ss)
next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply-
  apply(frul  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (smt(verit) prem(1) prem(2) ss)
qed
next
  case spec-resolveO note srO3 = spec-resolveO
  then have r4:is-Output (prog ! pcOf (last cfgs4)) using  $\Delta 2$ -implies[OF
 $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  srO3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  srO3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case spec-normal

```

```

    then show ?thesis using r4 by auto
  next
    case spec-mispred
    then show ?thesis using r4 by auto
  next
    case spec-Fence
    then show ?thesis using r4 by auto
  next
    case spec-resolveI
    then show ?thesis using r4 by auto
  next
    case spec-resolve note srO4 = spec-resolve
    show ?thesis using stat Δ2 prem srO3 srO4
    unfolding ss lcfgs apply-
    apply(frul Δ2-implies) apply (simp add: Δ2-defs Δ1-defs)
    apply(rule oor3I3, simp add: Δ1-defs)
    by (smt(verit) prem(1) prem(2) ss)
  next
    case spec-resolveO note sr4 = spec-resolveO
    show ?thesis using stat Δ2 prem srO3 sr4
    unfolding ss lcfgs apply-
    apply(frul Δ2-implies) apply (simp add: Δ2-defs Δ1-defs)
    apply(rule oor3I3, simp add: Δ1-defs)
    by (smt(verit) prem(1) prem(2) ss)
qed
next
  case spec-resolveI note srI3 = spec-resolveI
  then have r4:is-getInput (prog ! pcOf (last cfgs4)) using Δ2-implies[OF
Δ2[unfolded ss]] unfolding pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using stat Δ2 srI3 unfolding ss by (simp add:
Δ2-defs)
  next
    case nonspec-mispred
    then show ?thesis using stat Δ2 srI3 unfolding ss by (simp add:
Δ2-defs)
  next
    case spec-normal
    then show ?thesis using r4 by auto
  next
    case spec-mispred
    then show ?thesis using r4 by auto
  next
    case spec-Fence
    then show ?thesis using r4 by auto
  next
    case spec-resolveO
    then show ?thesis using r4 by auto

```

```

next
  case spec-resolve note  $srO_4 = \text{spec-resolve}$ 
  show ?thesis using  $\Delta_2$  prem  $srI_3$   $srO_4$ 
  unfolding  $ss$  lcfgs apply-
  apply(frule  $\Delta_2$ -implies) apply (simp add:  $\Delta_2$ -defs  $\Delta_1$ -defs)
  apply(rule  $oor3I_3$ , simp add:  $\Delta_1$ -defs)
  by (smt(verit) prem(1) prem(2)  $ss$ )
next
  case spec-resolveI note  $sr_4 = \text{spec-resolveI}$ 
  show ?thesis using  $\Delta_2$  prem  $srI_3$   $sr_4$ 
  unfolding  $ss$  lcfgs apply-
  apply(frule  $\Delta_2$ -implies) apply (simp add:  $\Delta_2$ -defs  $\Delta_1$ -defs)
  apply(rule  $oor3I_3$ , simp add:  $\Delta_1$ -defs)
  by (smt(verit) prem(1) prem(2)  $ss$ )
qed
qed . . .
qed
qed

```

lemma *step3: unwindIntoCond* Δ_3 (*oor* Δ_3 Δ_1)

proof(*rule* *unwindIntoCond-simpleI*)

fix n w_1 w_2 ss_3 ss_4 *statA* ss_1 ss_2 *statO*

assume r : *reachO* ss_3 *reachO* ss_4 *reachV* ss_1 *reachV* ss_2

and Δ_3 : Δ_3 n w_1 w_2 ss_3 ss_4 *statA* ss_1 ss_2 *statO*

obtain $pstate_3$ cfg_3 $cfgs_3$ ibT_3 $ibUT_3$ ls_3 **where** ss_3 : $ss_3 = (pstate_3, cfg_3, cfgs_3, ibT_3, ibUT_3, ls_3)$

by (*cases* ss_3 , *auto*)

obtain $pstate_4$ cfg_4 $cfgs_4$ ibT_4 $ibUT_4$ ls_4 **where** ss_4 : $ss_4 = (pstate_4, cfg_4, cfgs_4, ibT_4, ibUT_4, ls_4)$

by (*cases* ss_4 , *auto*)

obtain cfg_1 ibT_1 $ibUT_1$ ls_1 **where** ss_1 : $ss_1 = (cfg_1, ibT_1, ibUT_1, ls_1)$

by (*cases* ss_1 , *auto*)

obtain cfg_2 ibT_2 $ibUT_2$ ls_2 **where** ss_2 : $ss_2 = (cfg_2, ibT_2, ibUT_2, ls_2)$

by (*cases* ss_2 , *auto*)

note $ss = ss_3$ ss_4 ss_1 ss_2

obtain pc_3 vs_3 $avst_3$ h_3 p_3 **where**

$lcfgs_3$: *last* $cfgs_3 = \text{Config } pc_3 (\text{State } (Vstore\ vs_3) avst_3\ h_3\ p_3)$

by (*cases* *last* $cfgs_3$) (*metis* *state.collapse* *vstore.collapse*)

obtain pc_4 vs_4 $avst_4$ h_4 p_4 **where**

$lcfgs_4$: *last* $cfgs_4 = \text{Config } pc_4 (\text{State } (Vstore\ vs_4) avst_4\ h_4\ p_4)$

by (*cases* *last* $cfgs_4$) (*metis* *state.collapse* *vstore.collapse*)

note $lcfgs = lcfgs_3$ $lcfgs_4$

obtain hh_3 **where** h_3 : $h_3 = \text{Heap } hh_3$ **by**(*cases* h_3 , *auto*)

obtain hh_4 **where** h_4 : $h_4 = \text{Heap } hh_4$ **by**(*cases* h_4 , *auto*)

```

note  $hh = h3\ h4$ 

have  $f1:\neg finalN\ ss1$ 
using  $\Delta3\ unfolding\ ss\ \Delta3-def$ 
apply  $clarsimp$ 
by( $frule\ common-implies,\ simp$ )

have  $f2:\neg finalN\ ss2$ 
using  $\Delta3\ unfolding\ ss\ \Delta3-def$ 
apply  $clarsimp$ 
by( $frule\ common-implies,\ simp$ )

have  $f3:\neg finalS\ ss3$ 
using  $\Delta3\ unfolding\ ss$ 
apply-apply( $frule\ \Delta3-implies$ )
using  $finalS-if-spec\ by\ force$ 

have  $f4:\neg finalS\ ss4$ 
using  $\Delta3\ unfolding\ ss$ 
apply-apply( $frule\ \Delta3-implies$ )
using  $finalS-if-spec\ by\ force$ 

note  $finals = f1\ f2\ f3\ f4$ 
show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
using  $finals\ by\ auto$ 

then show  $isIntO\ ss3 = isIntO\ ss4\ by\ simp$ 

then have  $lpc3:pcOf\ (last\ cfs3) = 6 \vee$ 
 $pcOf\ (last\ cfs3) = 7$ 
using  $\Delta3\ unfolding\ ss\ \Delta3-defs\ by\ simp$ 

have  $sec3[simp]:\neg isSecO\ ss3$ 
using  $\Delta3\ unfolding\ ss\ by\ (simp\ add:\ \Delta3-defs)$ 
have  $sec4[simp]:\neg isSecO\ ss4$ 
using  $\Delta3\ unfolding\ ss\ by\ (simp\ add:\ \Delta3-defs)$ 

have  $stat[simp]:\wedge s3'\ s4'\ statA'.\ statA' = sstatA'\ statA\ ss3\ ss4 \implies$ 
 $validTransO\ (ss3,\ s3') \implies validTransO\ (ss4,\ s4') \implies$ 
 $(statA = statA' \vee statO = Diff)$ 
subgoal for  $ss3'\ ss4'$ 
apply ( $cases\ ss3,\ cases\ ss4,\ cases\ ss1,\ cases\ ss2$ )
apply ( $cases\ ss3',\ cases\ ss4',\ clarsimp$ )
using  $\Delta3\ finals\ unfolding\ ss\ apply\ clarsimp$ 
apply( $simp-all\ add:\ \Delta3-defs\ sstatA'-def$ )
apply( $cases\ statO,\ simp-all$ ) by ( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ ) .

```

```

have vs3 xx = vs4 xx using Δ3 lcfgs unfolding ss Δ3-defs misSpecL1-def
apply clarsimp
  by (metis cfigs-Suc-zero config.sel(2) list.set-intros(1) state.sel(1) vstore.sel)

then have a1x:(array-loc aa1 (nat (vs4 xx)) avst4) =
  (array-loc aa1 (nat (vs3 xx)) avst3)
  using Δ3 lcfgs unfolding ss Δ3-defs array-loc-def misSpecL1-def apply
  clarsimp
  by (metis Zero-not-Suc config.sel(2) last-in-set list.size(3) state.sel(2))

have oor2-rule:∧ss3' ss4'. ss3 →S ss3' ⇒ ss4 →S ss4' ⇒
  (pcOf (last cfigs3) = 6 → oor Δ3 Δ1 ∞ 1 1 ss3' ss4' (sstatA'
  statA ss3 ss4) ss1 ss2 statO)
  ∧ (pcOf (last cfigs3) = 7 → oor Δ3 Δ1 ∞ 0 0 ss3' ss4' (sstatA'
  statA ss3 ss4) ss1 ss2 statO) ⇒
  ∃ w1' < w1. ∃ w2' < w2. oor Δ3 Δ1 ∞ w1' w2' ss3' ss4' (sstatA'
  statA ss3 ss4) ss1 ss2 statO
  subgoal for ss3' ss4' apply(cases ss3', cases ss4')
  subgoal for pstate3' cfig3' cfigs3' ib3' ls3'
  pstate4' cfig4' cfigs4' ib4' ls4'
  using lpc3 apply(elim disjE)

subgoal apply(intro exI[of - 1], intro conjI)
subgoal using Δ3 unfolding ss Δ3-defs apply clarify
  by (metis enat-ord-simps(4) infinity-ne-i1)
apply(intro exI[of - 1], rule conjI)
subgoal using Δ3 unfolding ss Δ3-defs apply clarify
  by (metis enat-ord-simps(4) infinity-ne-i1)
by simp

apply(intro exI[of - 0], intro conjI)
subgoal using Δ3 unfolding ss Δ3-defs by (clarify,metis zero-less-one)
apply(intro exI[of - 0], rule conjI)
subgoal using Δ3 unfolding ss Δ3-defs by (clarify,metis zero-less-one)
by simp . .

have cfigs3:cfigs3 ≠ [] using Δ3 misSpecL1-ne unfolding ss Δ3-defs by fast
have pstate3:pstate3 = pstate4 using Δ3[unfolded ss Δ3-defs] by auto
have pcs:pcOf (last cfigs4) = pcOf (last cfigs3) using Δ3-implies[OF Δ3[unfolded
  ss]] by auto

show react (oor Δ3 Δ1) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

  show match1 (oor Δ3 Δ1) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor Δ3 Δ1) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor Δ3 Δ1) w1 w2 ss3 ss4 statA ss1 ss2 statO

```

```

apply(rule match12-simpleI, simp-all, rule disjI1)
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
  subgoal for pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3'
    pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4'
  apply-apply(rule oor2-rule, assumption+, intro conjI impI)

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using stat Δ3 cfgs3 unfolding ss by (auto simp add:
Δ3-defs)
  next
  case nonspec-mispred
    then show ?thesis using stat Δ3 cfgs3 unfolding ss by (auto simp add:
Δ3-defs)
  next
  case spec-mispred
    then show ?thesis using stat Δ3 prem(6) unfolding ss by (auto simp
add: Δ3-defs)
  next
  case spec-Fence
    then show ?thesis using stat Δ3 prem(6) unfolding ss by (auto simp
add: Δ3-defs)
  next
  case spec-resolveI
    then show ?thesis using prem(6) pcs by auto
  next
  case spec-resolveO
    then show ?thesis using prem(6) pcs by auto
  next
  case spec-resolve note sr3 = spec-resolve
    then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) cfgs4 ≠ []
using Δ3-implies[OF Δ3[unfolded ss]] unfolding pstate3 by auto
    show ?thesis
    using prem(2)[unfolded ss prem(4,5), simplified] proof(cases rule:
stepS-cases)
      case nonspec-mispred then show ?thesis using r4 by auto
    next
      case spec-mispred then show ?thesis using r4 by auto
    next
      case spec-Fence then show ?thesis using r4 by auto
    next
      case nonspec-normal then show ?thesis using r4 by auto
    next
      case spec-normal then show ?thesis using r4 by auto
    next
      case spec-resolveO then show ?thesis using prem(6) pcs by auto
    next
      case spec-resolveI then show ?thesis using prem(6) pcs by auto

```

```

next
  case spec-resolve note sr4 = spec-resolve
  have butlast:butlast cfigs3 = [] butlast cfigs4 = [] using Δ3-implies[OF
Δ3[unfolded ss]] by auto
  then show ?thesis
  apply(intro oorI2)
  unfolding ss Δ1-def prem(4,5) apply– apply(clarify,intro conjI)
  subgoal using Δ3 lcfgs prem(1,2) sr3 sr4 stat unfolding ss hh
  apply– by(simp add: Δ3-defs Δ1-defs butlast, metis)
  subgoal using stat Δ3 lcfgs prem(4,5) sr3 sr4 unfolding ss hh
  apply– apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs)
  subgoal using stat Δ3 lcfgs prem(4,5) sr3 sr4 unfolding ss hh
  apply– apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs)
  subgoal using stat Δ3 lcfgs prem(4,5) sr3 sr4 unfolding ss hh
  apply– apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs) .
qed
next
case spec-normal note sn3 = spec-normal
show ?thesis
using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat Δ3 lcfgs sn3 unfolding ss by (simp add:
Δ3-defs)
  next
  case nonspec-mispred
  then show ?thesis using stat Δ3 lcfgs sn3 unfolding ss by (simp add:
Δ3-defs)
  next
  case spec-mispred
  then show ?thesis using stat Δ3 lcfgs sn3 unfolding ss by (simp add:
Δ3-defs, metis config.sel(1) last-map)
  next
  case spec-Fence
  then show ?thesis using stat Δ3 lcfgs sn3 unfolding ss
  by (simp add: Δ3-defs, metis config.sel(1) last-map)
  next
  case spec-resolve
  then show ?thesis using stat Δ3 lcfgs sn3 unfolding ss by (simp add:
Δ3-defs)
  next
  case spec-resolveI
  then show ?thesis using prem(6) pcs by auto
  next
  case spec-resolveO
  then show ?thesis using prem(6) pcs by auto
  next
  case spec-normal note sn4 = spec-normal
  show ?thesis
  apply(intro oorI1)

```

```

      unfolding ss  $\Delta 3$ -def prem(4,5) apply clarify apply- apply(intro
conjI)
      subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
      using cases-12[of pc3] apply simp apply(elim disjE)
      apply simp-all by (metis config.sel(2) last-in-set state.sel(2) Dist-ignore
a1x )
      subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss prem(4,5)
hh
      apply- apply(frule  $\Delta 3$ -implies) apply(simp-all add:  $\Delta 3$ -defs)
      using cases-12[of pc3] apply simp apply(elim disjE)
      apply simp-all
      by (metis config.collapse config.inject last-in-set state.sel(1) vstore.sel)
      subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss prem(4,5)
hh
      apply- apply(frule  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs)
      subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
      using cases-12[of pc3] apply simp apply(elim disjE)
      by simp-all
      subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
      using cases-12[of pc3] apply (simp add: array-loc-def) apply(elim
disjE)
      by (simp-all add: array-loc-def)
      subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
      using cases-12[of pc3] apply (simp add: array-loc-def) apply(elim
disjE)
      by (simp-all add: array-loc-def)
      subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs) .
      qed
    qed

  subgoal premises prem using prem(1)[unfolded ss prem(4)]
  proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using cfigs3 by auto
  next
    case nonspec-mispred
    then show ?thesis using cfigs3 by auto
  next
    case spec-mispred
    then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
  next
    case spec-normal
    then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp

```

```

add:  $\Delta 3$ -defs)
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfs4) cfs4  $\neq$  []
using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis
    using prem(2)[unfolded ss prem(4,5), simplified] proof(cases rule:
stepS-cases)
      case nonspec-mispred then show ?thesis using r4 by auto
      next
      case spec-mispred then show ?thesis using r4 by auto
      next
      case spec-Fence then show ?thesis using r4 by auto
      next
      case nonspec-normal then show ?thesis using r4 by auto
      next
      case spec-normal then show ?thesis using r4 by auto
      next
      case spec-resolveO then show ?thesis using prem(6) pcs by auto
      next
      case spec-resolveI then show ?thesis using prem(6) pcs by auto
      next
      case spec-resolve note sr4 = spec-resolve
      have butlast:butlast cfs3 = [] butlast cfs4 = [] using  $\Delta 3$ -implies[OF
 $\Delta 3$ [unfolded ss]] by auto
      then show ?thesis
      apply(intro oorI2)
      unfolding ss  $\Delta 1$ -def prem(4,5) apply- apply(clarify,intro conjI)
      subgoal using  $\Delta 3$  lcfgs prem(1,2) sr3 sr4 stat unfolding ss hh
      apply- by(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs butlast, metis)
      subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sr3 sr4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sr3 sr4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sr3 sr4 unfolding ss hh
      apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs) .
      qed
    next
    case spec-resolveI then show ?thesis using prem(6) pcs by auto
    next
    case spec-resolveO then show ?thesis using prem(6) pcs by auto
    next
    case spec-Fence note sf3 = spec-Fence
    show ?thesis
      using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
        case nonspec-normal
        then show ?thesis using stat  $\Delta 3$  lcfgs sf3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
      next

```

```

      case nonspec-mispred
    then show ?thesis using stat  $\Delta 3$  lcfgs sf3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
  next
    case spec-mispred
  then show ?thesis using stat  $\Delta 3$  lcfgs sf3 unfolding ss
  apply (simp add:  $\Delta 3$ -defs)
  by (metis com.disc config.sel(1) last-map)
  next
    case spec-resolve
  then show ?thesis using stat  $\Delta 3$  lcfgs sf3 unfolding ss
  by (simp add:  $\Delta 3$ -defs)
  next
    case spec-normal
  then show ?thesis using stat  $\Delta 3$  lcfgs sf3 unfolding ss
  apply (simp add:  $\Delta 3$ -defs)
  by (metis last-map local.spec-Fence(3) local.spec-normal(1) lo-
cal.spec-normal(4))
  next
    case spec-resolveI then show ?thesis using prem(6) pcs by auto
  next
    case spec-resolveO then show ?thesis using prem(6) pcs by auto
  next
    case spec-Fence note sf4 = spec-Fence
  show ?thesis
  apply(intro oorI2)
  unfolding ss  $\Delta 1$ -def prem(4,5) apply- apply(clarify,intro conjI)
  subgoal using  $\Delta 3$  lcfgs prem(1,2) sf3 sf4 unfolding ss hh
  apply- by(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs, metis ss stat validTransO.simps)

  subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sf3 sf4 unfolding ss hh
  apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
  subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sf3 sf4 unfolding ss hh
  apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
  subgoal using stat  $\Delta 3$  lcfgs prem(4,5) sf3 sf4 unfolding ss hh
  apply- apply(frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs) .
  qed
  qed . . .
  qed
  qed

```

```

lemma step4: unwindIntoCond  $\Delta 1'$   $\Delta 1$ 
proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 1'$ :  $\Delta 1'$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

```

```

obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
by (cases ss3, auto)
obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,
ibT4, ibUT4, ls4)
by (cases ss4, auto)
obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
by (cases ss1, auto)
obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
by (cases ss2, auto)
note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
note hh = h3 h4

have f1: $\neg$ finalN ss1
using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -def
apply clarsimp
by(frule common-implies, simp)

have f2: $\neg$ finalN ss2
using  $\Delta 1'$  unfolding ss  $\Delta 1'$ -def
apply clarsimp
by(frule common-implies, simp)

have f3: $\neg$ finalS ss3
using  $\Delta 1'$  unfolding ss
apply—apply(frule  $\Delta 1'$ -implies)
by (simp add: finalS-while-spec)

have f4: $\neg$ finalS ss4
using  $\Delta 1'$  unfolding ss
apply—apply(frule  $\Delta 1'$ -implies)
by (simp add: finalS-while-spec)

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
using finals by auto

```

then show $isIntO\ ss3 = isIntO\ ss4$ **by** *simp*

have *match12-aux*:

```
( $\wedge s1' s2' statA'$ 
   $statA' = sstatA' statA\ ss3\ ss4 \implies$ 
   $validTransO\ (ss3, s1') \implies$ 
   $validTransO\ (ss4, s2') \implies$ 
   $Opt.eqAct\ ss3\ ss4 \implies$ 
  ( $\neg isSecO\ ss3 \wedge \neg isSecO\ ss4 \wedge$ 
    $(statA = statA' \vee statO = Diff) \wedge$ 
    $\Delta1 \infty 1\ 1\ s1'\ s2'\ statA'\ ss1\ ss2\ statO$ )
 $\implies match12\ \Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
apply(rule match12-simpleI, rule disjI1)
```

```
apply(rule exI[of - 1], rule conjI)
  subgoal using  $\Delta1'$  unfolding ss  $\Delta1'$ -defs apply clarify
  by(metis enat-ord-simps(4) infinity-ne-i1)
apply(rule exI[of - 1], rule conjI)
  subgoal using  $\Delta1'$  unfolding ss  $\Delta1'$ -defs apply clarify
  by(metis enat-ord-simps(4) infinity-ne-i1)
by auto
```

show *react* $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$

unfolding *react-def* **proof**(*intro conjI*)

```
show match1  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding match1-def by (simp add: finalS-def final-def)
show match2  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding match2-def by (simp add: finalS-def final-def)
show match12  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
proof(rule match12-aux, intro conjI)
  fix  $ss3'\ ss4'\ statA'$ 
  assume  $statA'$ :  $statA' = sstatA' statA\ ss3\ ss4$ 
  and  $v$ :  $validTransO\ (ss3, ss3')\ validTransO\ (ss4, ss4')$ 
  and  $sa$ :  $Opt.eqAct\ ss3\ ss4$ 
  note  $v3 = v(1)$  note  $v4 = v(2)$ 

  obtain  $pstate3'\ cfg3'\ cfs3'\ ibT3'\ ibUT3'\ ls3'$  where  $ss3'$ :  $ss3' = (pstate3',$ 
 $cfg3', cfs3', ibT3', ibUT3', ls3')$ 
  by (cases ss3', auto)
  obtain  $pstate4'\ cfg4'\ cfs4'\ ibT4'\ ibUT4'\ ls4'$  where  $ss4'$ :  $ss4' = (pstate4',$ 
 $cfg4', cfs4', ibT4', ibUT4', ls4')$ 
  by (cases ss4', auto)
  note  $ss = ss\ ss3'\ ss4'$ 

  obtain  $hh3$  where  $h3$ :  $h3 = Heap\ hh3$  by(cases h3, auto)
  obtain  $hh4$  where  $h4$ :  $h4 = Heap\ hh4$  by(cases h4, auto)
  note  $hh = h3\ h4$ 
```

```

show  $\neg$  isSecO ss3
using v sa  $\Delta 1'$  unfolding ss by (simp add:  $\Delta 1'$ -defs, linarith)

show  $\neg$  isSecO ss4
using v sa  $\Delta 1'$  unfolding ss by (simp add:  $\Delta 1'$ -defs, linarith)

show stat: statA = statA'  $\vee$  statO = Diff

using v sa  $\Delta 1'$ 
apply (cases ss3, cases ss4, cases ss1, cases ss2)
  apply(cases ss3', cases ss4', clarsimp)
using v sa  $\Delta 1'$  unfolding ss statA' apply clarsimp
apply(simp-all add:  $\Delta 1'$ -defs sstatA'-def)
apply(cases statO, simp-all)
apply(cases statA, simp-all add: newStat-EqI)
unfolding finalS-def final-def
using One-nat-def less-numeral-extra(4)
  less-one list.size(3) map-is-Nil-conv
by (smt (verit) status.exhaust newStat-diff)

have pcs:pcOf (last cfgs4) = pcOf (last cfgs3) using  $\Delta 1'$ [unfolded ss  $\Delta 1'$ -defs]
by auto

show  $\Delta 1 \infty 1 1$  ss3' ss4' statA' ss1 ss2 statO
  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
    case nonspec-mispred
      then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
  next
    case spec-Fence
      then show ?thesis using sa  $\Delta 1'$  unfolding ss
        apply (simp add:  $\Delta 1'$ -defs, clarify, elim disjE)
        by (simp-all add:  $\Delta 1'$ -defs  $\Delta 1'$ -defs)
  next
    case spec-mispred
      then show ?thesis using sa  $\Delta 1'$  unfolding ss
        apply (simp add:  $\Delta 1'$ -defs, clarify, elim disjE)
        by (simp-all add:  $\Delta 1'$ -defs  $\Delta 1'$ -defs)
  next
    case spec-normal note sn3 = spec-normal
      show ?thesis using  $\Delta 1'$  sn3(6,7) pcs is-Output-1 unfolding ss
        apply (simp add:  $\Delta 1'$ -defs, clarify)

```

```

    apply(erule disjE)
  by (metis is-getInput-pcOf numeral-less-iff semiring-norm(77,78))+
next
  case spec-resolveI note sr3 = spec-resolveI
show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
  case spec-mispred
  then show ?thesis using sr3 pcs by auto
  next
  case spec-normal
  then show ?thesis using sr3 pcs by auto
  next
  case spec-Fence
  then show ?thesis using sr3 pcs by auto
  next
  case spec-resolveO
  then show ?thesis using sr3 pcs by auto
  next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
    nat-le-linear numeral-le-iff semiring-norm(68,69,72)
    length-1-butlast length-map in-set-butlastD)
  next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
    nat-le-linear numeral-le-iff semiring-norm(68,69,72)
    length-1-butlast length-map in-set-butlastD)
qed
next
  case spec-resolveO note sr3 = spec-resolveO
show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
  case spec-mispred

```

```

    then show ?thesis using sr3 pcs by auto
next
  case spec-normal
  then show ?thesis using sr3 pcs by auto
next
  case spec-Fence
  then show ?thesis using sr3 pcs by auto
next
  case spec-resolveI
  then show ?thesis using sr3 pcs by auto
next
  case spec-resolveO note sr4 = spec-resolveO
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
        nat-le-linear numeral-le-iff semiring-norm(68,69,72)
        length-1-butlast length-map in-set-butlastD)
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
        nat-le-linear numeral-le-iff semiring-norm(68,69,72)
        length-1-butlast length-map in-set-butlastD)
qed
next
  case spec-resolve note sr3 = spec-resolve
  show ?thesis using v4 [unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
  next
    case spec-mispred
    then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)

next
  case spec-normal
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
next
  case spec-Fence
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)

next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis

```

```

using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
      nat-le-linear numeral-le-iff semiring-norm(68,69,72)
      length-1-butlast length-map in-set-butlastD)
next
case spec-resolveO note sr4 = spec-resolveO
show ?thesis
using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
      nat-le-linear numeral-le-iff semiring-norm(68,69,72)
      length-1-butlast length-map in-set-butlastD)
next
case spec-resolve note sr4 = spec-resolve
show ?thesis
using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
      nat-le-linear numeral-le-iff semiring-norm(68,69,72)
      length-1-butlast length-map in-set-butlastD)
qed
qed
qed
qed
qed

```

```

lemma step5: unwindIntoCond  $\Delta 2'$   $\Delta 2$ 
proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 2'$ :  $\Delta 2'$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

  obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where

```

```

cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
note hh = h3 h4

have f1: $\neg$ finalN ss1
using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -def
apply clarsimp
by(frule common-implies, simp)

have f2: $\neg$ finalN ss2
using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -def
apply clarsimp
by(frule common-implies, simp)

have f3: $\neg$ finalS ss3
using  $\Delta 2'$  unfolding ss
apply—apply(frule  $\Delta 2'$ -implies)
using finalS-while-spec-L2 by force

have f4: $\neg$ finalS ss4
using  $\Delta 2'$  unfolding ss
apply—apply(frule  $\Delta 2'$ -implies)
using finalS-while-spec-L2 by force

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

have sec3[simp]: $\neg$  isSecO ss3
using  $\Delta 2'$  unfolding ss by (simp add:  $\Delta 2'$ -defs)
have sec4[simp]: $\neg$  isSecO ss4
using  $\Delta 2'$  unfolding ss by (simp add:  $\Delta 2'$ -defs)

have stat[simp]: $\wedge$  s3' s4' statA'. statA' = sstatA' statA ss3 ss4  $\implies$ 
validTransO (ss3, s3')  $\implies$  validTransO (ss4, s4')  $\implies$ 
(statA = statA'  $\vee$  statO = Diff)
subgoal for ss3' ss4'
apply (cases ss3, cases ss4, cases ss1, cases ss2)

```

```

apply(cases ss3', cases ss4', clarsimp)
using  $\Delta 2'$  finals unfolding ss apply clarsimp
apply(simp-all add:  $\Delta 2'$ -defs sstatA'-def)
apply(cases statO, simp-all) by (cases statA, simp-all add: newStat-EqI) .

```

```

have match12-aux:
( $\wedge$ pstate3' cfg3' cfgs3' ib3' ibUT3' ls3'
 pstate4' cfg4' cfgs4' ib4' ibUT4' ls4' statA'.
 (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)  $\rightarrow$ S (pstate3', cfg3', cfgs3', ib3',
 ibUT3', ls3')  $\implies$ 
 (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)  $\rightarrow$ S (pstate4', cfg4', cfgs4', ib4',
 ibUT4', ls4')  $\implies$ 
 Opt.eqAct ss3 ss4  $\implies$  statA' = sstatA' statA ss3 ss4  $\implies$ 
 ( $\Delta 2 \infty 1 1$  (pstate3', cfg3', cfgs3', ib3', ibUT3', ls3') (pstate4', cfg4', cfgs4',
 ib4', ibUT4', ls4') statA' ss1 ss2 statO))
  $\implies$  match12  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-simpleI, simp-all, rule disjI1)

```

```

apply(rule exI[of - 1], rule conjI)
subgoal using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
apply(rule exI[of - 1], rule conjI)
subgoal using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
subgoal for pstate3' cfg3' cfgs3' ib3' ibUT3' ls3'
 pstate4' cfg4' cfgs4' ib4' ibUT4' ls4'
using ss3 ss4 by blast . .

```

```

have pstate3:pstate3 = pstate4 using  $\Delta 2'$ [unfolded ss  $\Delta 2'$ -defs] by fast

```

```

show react  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

```

```

show match1  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-aux)

```

```

subgoal premises prem using prem(1)[unfolded ss]
proof(cases rule: stepS-cases)
case nonspec-normal
then show ?thesis using stat  $\Delta 2'$  unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
next
case nonspec-mispred

```

```

      then show ?thesis using stat  $\Delta 2'$  unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
    next
      case spec-mispred
      then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
    next
      case spec-normal
      then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
    next
      case spec-Fence
      then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
    next
      case spec-resolve note sr3 = spec-resolve
      have r4: resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sr3  $\Delta 2'$ [unfolded
ss  $\Delta 2'$ -defs] pstate3 by auto
      show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)
        case nonspec-normal
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
      next
        case nonspec-mispred
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
      next
        case spec-mispred
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
      next
        case spec-normal
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
      next
        case spec-Fence
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
      next
        case spec-resolveI note sr4 = spec-resolveI(1,3-) r4
        show ?thesis
        using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
        apply (simp add:  $\Delta 2'$ -defs  $\Delta 2'$ -defs, elim conjE)
        apply (intro conjI)
        apply (metis last-map map-butlast map-is-Nil-conv)
        apply (metis image-subset-iff in-set-butlastD)
        apply (metis) apply (metis) apply (metis in-set-butlastD)
        apply (metis in-set-butlastD) apply (metis in-set-butlastD)
        apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)

```

```

apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
        list.simps(8) map-L2 map-butlast not-Cons-self2)
subgoal premises p using p(65,66,67) length2-butlast by fastforce
subgoal premises p using p(65,66,67) length2-butlast by fastforce .
next
case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
show ?thesis
using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
apply(intro conjI)
apply (metis last-map map-butlast map-is-Nil-conv)
apply (metis image-subset-iff in-set-butlastD)
apply(metis) apply(metis) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
        list.simps(8) map-L2 map-butlast not-Cons-self2)
subgoal premises p using p(65,66,67) length2-butlast by fastforce
subgoal premises p using p(65,66,67) length2-butlast by fastforce .
next
case spec-resolve note sr4 = spec-resolve
show ?thesis
using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
apply(intro conjI)
apply (metis last-map map-butlast map-is-Nil-conv)
apply (metis image-subset-iff in-set-butlastD)
apply(metis) apply(metis) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
        list.simps(8) map-L2 map-butlast not-Cons-self2)
subgoal premises p using p(65,66,67) length2-butlast by fastforce
subgoal premises p using p(65,66,67) length2-butlast by fastforce .
qed
next
case spec-resolveO note sr3 = spec-resolveO
have r4:is-Output (prog ! pcOf (last cfigs4)) using sr3  $\Delta 2'$ -implies[OF
 $\Delta 2'$ [unfolded ss]] by auto
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
case nonspec-normal
then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
next
case nonspec-mispred
then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:

```

```

Δ2'-defs)
  next
  case spec-mispred
  then show ?thesis using r4 by auto
  next
  case spec-normal
  then show ?thesis using r4 by auto
  next
  case spec-Fence
  then show ?thesis using r4 by auto
  next
  case spec-resolveI
  then show ?thesis using r4 by auto
  next
  case spec-resolveO note sr4 = spec-resolveO(1,3-) r4
  show ?thesis
  using stat Δ2' prem sr3 sr4 unfolding ss
  apply(simp add: Δ2'-defs Δ2-defs, elim conjE)
  apply(intro conjI)
  apply (metis last-map map-butlast map-is-Nil-conv)
  apply (metis image-subset-iff in-set-butlastD)
  apply(metis) apply(metis) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (smt (verit, del-Insts) butlast.simps(2) last-ConsL last-map
    list.simps(8) map-L2 map-butlast not-Cons-self2)
  subgoal premises p using p(68-) length2-butlast by fastforce
  subgoal premises p using p(68-) length2-butlast by fastforce .
  next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  using stat Δ2' prem sr3 sr4 unfolding ss
  apply(simp add: Δ2'-defs Δ2-defs, elim conjE)
  apply(intro conjI)
  apply (metis last-map map-butlast map-is-Nil-conv)
  apply (metis image-subset-iff in-set-butlastD)
  apply(metis) apply(metis) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (smt (verit, del-Insts) butlast.simps(2) last-ConsL last-map
    list.simps(8) map-L2 map-butlast not-Cons-self2)
  subgoal premises p using p(65,66,67) length2-butlast by fastforce
  subgoal premises p using p(65,66,67) length2-butlast by fastforce .
  qed
  next
  case spec-resolveI note sr3 = spec-resolveI
  have r4:is-getInput (prog ! pcOf (last cfigs4)) using sr3 Δ2'-implies[OF

```

```

 $\Delta 2'$ [unfolded ss] by auto
  show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
    case nonspec-normal
      then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
    next
      case nonspec-mispred
        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
    next
      case spec-mispred
        then show ?thesis using r4 by auto
    next
      case spec-normal
        then show ?thesis using r4 by auto
    next
      case spec-Fence
        then show ?thesis using r4 by auto
    next
      case spec-resolveO
        then show ?thesis using r4 by auto
    next
      case spec-resolveI note sr4 = spec-resolveI(1,3-) r4
      show ?thesis
      using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
      apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
      apply(intro conjI)
      apply (metis last-map map-butlast map-is-Nil-conv)
      apply (metis image-subset-iff in-set-butlastD)
      apply(metis) apply(metis) apply (metis in-set-butlastD)
      apply (metis in-set-butlastD) apply (metis in-set-butlastD)
      apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
      apply (metis in-set-butlastD) apply (metis in-set-butlastD)
      apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
        list.simps(8) map-L2 map-butlast not-Cons-self2)
      subgoal premises p using p(68-) length2-butlast by fastforce
      subgoal premises p using p(68-) length2-butlast by fastforce .
    next
      case spec-resolve note sr4 = spec-resolve
      show ?thesis
      using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
      apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
      apply(intro conjI)
      apply (metis last-map map-butlast map-is-Nil-conv)
      apply (metis image-subset-iff in-set-butlastD)
      apply(metis) apply(metis) apply (metis in-set-butlastD)
      apply (metis in-set-butlastD) apply (metis in-set-butlastD)
      apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
      apply (metis in-set-butlastD) apply (metis in-set-butlastD)

```

```

    apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
           list.simps(8) map-L2 map-butlast not-Cons-self2)
    subgoal premises p using p(65,66,67) length2-butlast by fastforce
    subgoal premises p using p(65,66,67) length2-butlast by fastforce .
qed

qed .
qed
qed

lemma step: unwindIntoCond  $\Delta e$   $\Delta e$ 
proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta e$ :  $\Delta e$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

  obtain pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
  note hh = h3 h4

  show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
  using  $\Delta e$  Opt.final-def finalS-def stepS-endPC endPC-def finalB-endPC
  unfolding  $\Delta e$ -defs ss by clarsimp

  then show isIntO ss3 = isIntO ss4 by simp

  show react  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO

```

```

unfolding react-def proof(intro conjI)

  show match1  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
  apply(rule match12-simpleI) using  $\Delta e$  unfolding ss apply (simp add:  $\Delta e$ -defs)
  by (simp add: stepS-endPC)
qed
qed

```

lemmas *theConds* = *step0 step1 step2 step3 step4 step5 step6*

proposition *lrsecure*

proof –

```

define m where m: m  $\equiv$  (7::nat)
define  $\Delta s$  where  $\Delta s$ :  $\Delta s$   $\equiv$   $\lambda i::nat.$ 
  if i = 0 then  $\Delta 0$ 
  else if i = 1 then  $\Delta 1$ 
  else if i = 2 then  $\Delta 2$ 
  else if i = 3 then  $\Delta 3$ 
  else if i = 4 then  $\Delta 1'$ 
  else if i = 5 then  $\Delta 2'$ 
  else  $\Delta e$ 
define next where next: next  $\equiv$   $\lambda i::nat.$ 
  if i = 0 then {0,1::nat}
  else if i = 1 then {1,4,2,3,6}
  else if i = 2 then {2,5,1}
  else if i = 3 then {3,1}
  else if i = 4 then {1}
  else if i = 5 then {2}
  else {6}
show ?thesis apply(rule distrib-unwind-lrsecure[of m next  $\Delta s$ ])
  subgoal unfolding m by auto
  subgoal unfolding next m by auto
  subgoal using init unfolding  $\Delta s$  by auto
  subgoal
    unfolding m next  $\Delta s$  apply (simp split: if-splits)
    using theConds
    unfolding oor-def oor3-def oor4-def oor5-def by auto .
qed

```

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  $\geq$  0 by auto
```

```
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  $\neq$  aa2 aa1  $\neq$  vv aa1  $\neq$  xx aa1  $\neq$  yy aa1  $\neq$  tt aa1  $\neq$  ffile
```

```
aa2  $\neq$  aa1 aa2  $\neq$  vv aa2  $\neq$  xx aa2  $\neq$  yy aa2  $\neq$  tt aa2  $\neq$  ffile
```

```
vv  $\neq$  aa1 vv  $\neq$  aa2 vv  $\neq$  xx vv  $\neq$  yy vv  $\neq$  tt vv  $\neq$  ffile
```

```
xx  $\neq$  aa1 xx  $\neq$  aa2 xx  $\neq$  vv xx  $\neq$  yy xx  $\neq$  tt xx  $\neq$  ffile
```

```
tt  $\neq$  aa1 tt  $\neq$  aa2 tt  $\neq$  vv tt  $\neq$  yy tt  $\neq$  xx tt  $\neq$  ffile
```

```
yy  $\neq$  aa1 yy  $\neq$  aa2 yy  $\neq$  vv yy  $\neq$  xx yy  $\neq$  tt yy  $\neq$  ffile
```

```
ffile  $\neq$  aa1 ffile  $\neq$  aa2 ffile  $\neq$  vv ffile  $\neq$  xx ffile  $\neq$  tt ffile  $\neq$  yy
```

```
unfolding vvars-defs by auto
```

```
consts size-aa1 :: nat
```

```
consts size-aa2 :: nat
```

```
fun initAvstore :: avstore  $\Rightarrow$  bool where
```

```
initAvstore (Avstore as) = (as aa1 = (0, size-aa1)  $\wedge$  as aa2 = (size-aa1, size-aa2))
```

```
fun istate :: state  $\Rightarrow$  bool where
```

```
istate s = (initAvstore (getAvstore s))
```

```
definition prog  $\equiv$ 
```

```
[
```

```
   $\emptyset$  Start ,
```

```
   $\not\#$  tt ::= (N 0),
```

```
   $\not\#$  xx ::= (N 1),
```

```
   $\not\#$  IfJump (Not (Eq (V xx) (N 0))) 4 13 ,
```

```
   $\not\#$  Input U xx ,
```

```
   $\not\#$  Input T yy ,
```

```
   $\not\#$  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 $cases-14$: $(i::pcounter) = 0 \vee i = 1 \vee i = 2 \vee i = 3 \vee i = 4 \vee i = 5 \vee$
 $i = 6 \vee i = 7 \vee i = 8 \vee i = 9 \vee i = 10 \vee i = 11 \vee i = 12 \vee i = 13 \vee i = 14$
 $\vee i > 14$

apply(cases i, simp-all)

subgoal for i **apply**(cases i, simp-all)

lemma *getInput-not10*[simp]: \neg *is-getInput* (prog ! 10) **by**(simp add: prog-def)
lemma *Output-not10*[simp]: \neg *is-Output* (prog ! 10) **by**(simp add: prog-def)

lemma *getInput-not12*[simp]: \neg *is-getInput* (prog ! 12) **by**(simp add: prog-def)
lemma *Output-not12*[simp]: \neg *is-Output* (prog ! 12) **by**(simp add: prog-def)

lemma *fence*[simp]:prog ! 9 = Fence **by**(simp add: prog-def)

lemma *nfence*[simp]:prog ! 7 \neq Fence **by**(simp add: prog-def)

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*
by (standard, simp add: prog-def)

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
llist \times loc set \Rightarrow bool (**infix** $\langle \rightarrow N \rangle$ 55)
where $x \rightarrow N y == \text{stepN } x y$

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* = 14

unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getInput-pcOf[simp]*: *pcOf cfg* < 14 ⇒ *is-getInput (prog!(pcOf cfg))*
 ⇔ *pcOf cfg* = 4 ∨ *pcOf cfg* = 5

using *cases-14[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *is-Output-pcOf[simp]*: *pcOf cfg* < 14 ⇒ *is-Output (prog!(pcOf cfg))* ⇔
 (*pcOf cfg* = 8 ∨ *pcOf cfg* = 11 ∨ *pcOf cfg* = 13)

using *cases-14[of pcOf cfg]* **by** (*auto simp: prog-def*)

lemma *is-getInput1[simp]*: *is-getInput (prog ! 4)* **unfolding** *prog-def* **by** *auto*

lemma *is-Output-T*: *is-Output (prog ! 8)*

unfolding *is-Output-def prog-def* **by** *auto*

lemma *is-Output*: *is-Output (prog ! 11)*

unfolding *is-Output-def prog-def* **by** *auto*

lemma *is-Output-1*: *is-Output (prog ! 13)*

unfolding *is-Output-def prog-def* **by** *auto*

lemma *isSecV-pcOf[simp]*:

isSecV (cfg, ibT, ibUT, ls) ⇔ ¬*finalB (cfg, ibT, ibUT)*

using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf[simp]*:

isSecO (pstate, cfg, cfs, ibT, ibUT, ls) ⇔

¬*finalS (pstate, cfg, cfs, ibT, ibUT, ls)* ∧ *cfs* = []

using *isSecO-def* **by** *simp*

lemma *getActV-pcOf[simp]*:

```

pcOf cfg < 14 ==>
  getActV (cfg,ibT,ibUT,ls) =
    (if pcOf cfg = 4 then lhd ibUT
     else if pcOf cfg = 5 then lhd ibT
     else ⊥)
apply(subst getActV-simps) unfolding prog-def
apply simp
using getActV-simps not-is-getInput-getActV prog-def by auto

lemma getObsV-pcOf[simp]:
pcOf cfg < 14 ==>
  getObsV (cfg,ibT,ibUT,ls) =
    (if pcOf cfg = 11 ∨ pcOf cfg = 13 then
      (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
      else ⊥
    )
apply(subst getObsV-simps)
apply (simp add: prog-def)
unfolding getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def One-nat-def

using cases-14[of pcOf cfg] by auto

lemma getActO-pcOf[simp]:
pcOf cfg < 12 ==>
  getActO (pstate,cfg,cfgs,ibT,ibUT,ls) =
    (if cfgs = [] then
      (if pcOf cfg = 4 then lhd ibUT
       else if pcOf cfg = 5 then lhd ibT
       else ⊥) else ⊥)
apply(subst getActO-simps)
apply(cases cfgs, auto)
unfolding prog-def apply simp
apply(cases pcOf cfg = 4, auto)
using getActV-simps getActV-pcOf prog-def by simp

lemma getObsO-pcOf[simp]:
pcOf cfg < 14 ==>
  getObsO (pstate,cfg,cfgs,ibT,ibUT,ls) =
    (if (pcOf cfg = 11 ∨ pcOf cfg = 13) ∧ cfgs = [] then
      (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
      else ⊥
    )
apply(subst getObsO-simps)
apply(cases cfgs, auto)
using getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def
  One-nat-def
unfolding prog-def
using cases-14[of pcOf cfg]
by auto

```

lemma *getActTrustedInput*: $pc4 = 4 \implies pc3 = 4 \implies cfs3 = [] \implies cfs4 = []$
 \implies
 $getActO (pstate3, Config\ pc3 (State (Vstore\ vs3)\ avst3\ h3\ p3), [],\ ib3T,$
 $ib3UT, ls3) =$
 $getActO (pstate4, Config\ pc4 (State (Vstore\ vs4)\ avst4\ h4\ p4), [],\ ib4T,$
 $ib4UT, ls4)$
 $\implies lhd\ ib3UT = lhd\ ib4UT$
using *getActO-pcOf zero-less-numeral* **by** *auto*

lemma *getActUntrustedInput*: $pc4 = 5 \implies pc3 = 5 \implies cfs3 = [] \implies cfs4 = []$
 \implies
 $getActO (pstate3, Config\ pc3 (State (Vstore\ vs3)\ avst3\ h3\ p3), [],\ ib3T,$
 $ib3UT, ls3) =$
 $getActO (pstate4, Config\ pc4 (State (Vstore\ vs4)\ avst4\ h4\ p4), [],\ ib4T,$
 $ib4UT, ls4)$
 $\implies lhd\ ib3T = lhd\ ib4T$
using *getActO-pcOf zero-less-numeral* **by** *auto*

lemma *nextB-pc0[simp]*:
 $nextB (Config\ 0\ s, ibT, ibUT) = (Config\ 1\ s, ibT, ibUT)$
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc0[simp]*:
 $readLocs (Config\ 0\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1[simp]*:
 $nextB (Config\ 1 (State (Vstore\ vs)\ avst\ hh\ p), ibT, ibUT) =$
 $((Config\ 2 (State (Vstore (vs(tt := 0)))\ avst\ hh\ p), ibT, ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc1'[simp]*:
 $nextB (Config (Suc\ 0) (State (Vstore\ vs)\ avst\ hh\ p), ibT, ibUT) =$
 $((Config\ 2 (State (Vstore (vs(tt := 0)))\ avst\ hh\ p), ibT, ibUT)$
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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

unfolding *endPC-def readLocs-def unfolding prog-def by auto*

lemma *readLocs-pc1 [simp]*:

readLocs (Config (Suc 0) s) = {}

unfolding *endPC-def readLocs-def unfolding prog-def by auto*

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)

apply(subst nextB-Assign)

unfolding *endPC-def unfolding prog-def by auto*

lemma *readLocs-pc2 [simp]*:

readLocs (Config 2 s) = {}

unfolding *endPC-def readLocs-def unfolding prog-def by auto*

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

vs xx ≠ 0 ⇒

nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =

(Config 4 (State (Vstore vs) avst hh p), ibT, ibUT)

apply(subst nextB-IfTrue)

unfolding *endPC-def unfolding prog-def Eq-def by auto*

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

vs xx = 0 ⇒

nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =

(Config 13 (State (Vstore vs) avst hh p), ibT, ibUT)

apply(subst nextB-IfFalse)

unfolding *endPC-def unfolding prog-def Eq-def by auto*

lemma *nextB-pc3*:

nextB (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =

(Config (if vs xx ≠ 0 then 4 else 13) (State (Vstore vs) avst hh p), ibT, ibUT)

by *(cases vs xx = 0, auto)*

lemma *readLocs-pc3 [simp]*:

readLocs (Config 3 s) = {}

unfolding *endPC-def readLocs-def unfolding prog-def Eq-def by auto*

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

vs xx = 0 ⇒

nextM (Config 3 (State (Vstore vs) avst hh p), ibT, ibUT) =

(Config 4 (State (Vstore vs) avst hh p), ibT, ibUT)

apply(subst nextM-IfTrue)

unfolding *endPC-def unfolding prog-def Eq-def by auto*

lemma *nextM-pc3-else[simp]*:
 $vs\ xx \neq 0 \implies$
 $nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 13\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
apply(subst *nextM-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

lemma *nextM-pc3*:
 $nextM\ (Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ (if\ vs\ xx \neq 0\ then\ 13\ else\ 4)\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
by(cases *vs xx = 0*, *auto*)

lemma *nextB-pc4[simp]*:
 $ibUT \neq LNil \implies nextB\ (Config\ 4\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 5\ (State\ (Vstore\ (vs(xx := lhd\ ibUT))))\ avst\ hh\ p),\ ibT,\ ltl\ ibUT)$
apply(subst *nextB-getUntrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc4[simp]*:
 $readLocs\ (Config\ 4\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc5[simp]*:
 $ibT \neq LNil \implies nextB\ (Config\ 5\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 6\ (State\ (Vstore\ (vs(yy := lhd\ ibT))))\ avst\ hh\ p),\ ltl\ ibT,\ ibUT)$
apply(subst *nextB-getTrustedInput'*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc5[simp]*:
 $readLocs\ (Config\ 5\ s) = \{\}$
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc6-then[simp]*:
 $vs\ xx < int\ NN \implies$
 $nextB\ (Config\ 6\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT) =$
 $(Config\ 7\ (State\ (Vstore\ vs)\ avst\ hh\ p),\ ibT,\ ibUT)$
apply(subst *nextB-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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*)
apply(*subst nextB-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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)
by(*cases vs xx < int NN, auto*)

lemma *readLocs-pc6[simp]*:

readLocs (*Config 6 s*) = {}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def Eq-def* **by** *auto*

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*)
apply(*subst nextM-IfTrue*)
unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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*)
apply(*subst nextM-IfFalse*)
unfolding *endPC-def* **unfolding** *prog-def Eq-def* **by** *auto*

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)
by(*cases vs xx < int NN, auto*)

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc7[simp]*:

readLocs (*Config 7* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa1 (nat (vs xx))*
avst}
unfolding *endPC-def readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc8*[simp]:
readLocs (*Config 8 s*) = {}
unfolding *endPC-def* *readLocs-def*
unfolding *prog-def* **by** *auto*

lemma *nextB-pc9*[simp]:
nextB (*Config 9 s*, *ibT*, *ibUT*) = (*Config 10 s*, *ibT*, *ibUT*)
apply(*subst nextB-Start-Skip-Fence*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc9*[simp]:
readLocs (*Config 9 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc10*[simp]:
readLocs (*Config 10* (*State* (*Vstore vs*) *avst hh p*)) = {*array-loc aa2* (*nat (vs vv * 512)*) *avst*}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc11*[simp]:
nextB (*Config 11 s*, *ibT*, *ibUT*) = (*Config 12 s*, *ibT*, *ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc11*[simp]:
readLocs (*Config 11 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc12*[simp]:
nextB (*Config 12 s, ibT, ibUT*) = (*Config 3 s, ibT, ibUT*)
apply(*subst nextB-Jump*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc12*[simp]:
readLocs (*Config 12 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc13*[simp]:
nextB (*Config 13 s, ibT, ibUT*) =
(*Config 14 s, ibT, ibUT*)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc13*[simp]:
readLocs (*Config 13 s*) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *map-L1:length* *cfgs = Suc 0* \implies
pcOf (*last cfgs*) = *y* \implies *map pcOf cfgs* = [*y*]
by (*smt (verit, del-insts) Suc-length-conv cfgs-map last.simps*
length-0-conv map-eq-Cons-conv nth-Cons-0 numeral-2-eq-2)

lemma *map-L2:length* *cfgs = 2* \implies
pcOf (*cfgs ! 0*) = *x* \implies
pcOf (*last cfgs*) = *y* \implies *map pcOf cfgs* = [*x, y*]
by (*smt (verit) Suc-length-conv cfgs-map last.simps*
length-0-conv map-eq-Cons-conv nth-Cons-0 numeral-2-eq-2)

lemma *length2-butlast-eq:length* *cfgs = 2* \implies (*cfgs ! 0*) = *last (butlast cfgs)*
by (*cases cfgs, auto*)

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*)
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-14*[*of pc*] **apply** *safe*
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def)

subgoal apply(cases vs xx = 0)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply(cases vs xx = 0)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply(cases vs xx = 0)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply(cases vs xx = 0)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def Eq-def, auto) .
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def, metis llist.exhaust-sel)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def, metis llist.exhaust-sel)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def, metis llist.exhaust-sel)
subgoal apply simp apply(subst stepB.simps) **unfolding endPC-def**
by (simp add: prog-def, metis llist.exhaust-sel)

lemma *finalB-pc-iff*:

$pc \leq 14 \implies$

$finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$

$(pc = 14 \vee (pc = 4 \wedge ibUT = LNil) \vee (pc = 5 \wedge ibT = LNil))$

using *Prog.finalB-iff endPC finalB-pc-iff' order-le-less finalB-iff*

by *metis*

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

$pcOf\ cfg \leq 14 \implies$

$finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 14 \vee (pcOf\ cfg = 4 \wedge ibUT = LNil) \vee$

$(pcOf\ cfg = 5 \wedge ibT = LNil))$

using *config.collapse finalB-pc-iff by metis*

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)*

apply(*cases cfg*)

subgoal for *pc s* **apply**(*cases s*)

subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)

subgoal for *vs as h*

using *cases-14[of pc]* **apply**(*elim disjE*) **unfolding** *finalS-defs noMisSpec-def*

subgoal using *nonspec-normal[of [] Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT

Config 1 (State (Vstore vs) avst hh p)

ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]

using *is-If-pc by force*

subgoal apply(*frule nonspec-normal[of cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT

Config 2 (State (Vstore (vs(tt:= 0))) avst hh p)

ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal[of cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT

Config 3 (State (Vstore (vs(xx:= 1))) avst hh p)

ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]

prefer 7 subgoal by metis by simp-all

subgoal apply(*cases mispred pstate [?]*)

subgoal apply(*frule nonspec-mispred[of cfgs Config pc (State (Vstore vs) avst*

$hh\ p)$
 $(Vstore\ vs)\ avst\ hh\ p))]$
 $(State\ (Vstore\ vs)\ avst\ hh\ p)$
 $(State\ (Vstore\ vs)\ avst\ hh\ p)$
 $(State\ (Vstore\ vs)\ avst\ hh\ p)]$
 $avst\ hh\ p))\ ls]$
 $pstate\ update\ pstate\ [pcOf\ (Config\ pc\ (State$
 $ibT\ ibUT\ Config\ (if\ vs\ xx\ \neq\ 0\ then\ 4\ else\ 13)$
 $ibT\ ibUT\ Config\ (if\ vs\ xx\ \neq\ 0\ then\ 13\ else\ 4)$
 $ibT\ ibUT\ [Config\ (if\ vs\ xx\ \neq\ 0\ then\ 13\ else\ 4)$
 $ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore\ vs)$
prefer 9 subgoal by metis by (simp add: finalM-iff)+

subgoal apply(frule nonspec-normal[of cfigs Config pc (State (Vstore vs) avst
 $hh\ p)$
 $avst\ hh\ p)$
 $vs)\ avst\ hh\ p))\ ls]$
 $pstate\ pstate\ ibT\ ibUT$
 $Config\ (if\ vs\ xx\ \neq\ 0\ then\ 4\ else\ 13)\ (State\ (Vstore\ vs)$
 $ibT\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
prefer 7 subgoal by metis by simp-all .

subgoal apply(frule nonspec-normal[of cfigs Config pc (State (Vstore vs) avst
 $hh\ p)$
 $p)$
 $vs)\ avst\ hh\ p))\ ls]$
 $pstate\ pstate\ ibT\ ibUT$
 $Config\ 5\ (State\ (Vstore\ (vs(xx:=\ lhd\ ibUT)))\ avst\ hh$
 $ibT\ ltl\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
prefer 7 subgoal by metis by simp-all

subgoal apply(frule nonspec-normal[of cfigs Config pc (State (Vstore vs) avst
 $hh\ p)$
 $vs)\ avst\ hh\ p))\ ls]$
 $pstate\ pstate\ ibT\ ibUT$
 $Config\ 6\ (State\ (Vstore\ (vs(yy:=\ lhd\ ibT)))\ avst\ hh\ p)$
 $ltl\ ibT\ ibUT\ []\ ls\ \cup\ readLocs\ (Config\ pc\ (State\ (Vstore$
prefer 7 subgoal by metis by simp-all

subgoal apply(cases mispred pstate [6])
subgoal apply(frule nonspec-mispred[of cfigs Config pc (State (Vstore vs) avst
 $hh\ p)$
 $(Vstore\ vs)\ avst\ hh\ p))]$
 $12)\ (State\ (Vstore\ vs)\ avst\ hh\ p)$
 $7)\ (State\ (Vstore\ vs)\ avst\ hh\ p)$
 $pstate\ update\ pstate\ [pcOf\ (Config\ pc\ (State$
 $ibT\ ibUT\ Config\ (if\ vs\ xx\ <\ NN\ then\ 7\ else$
 $ibT\ ibUT\ Config\ (if\ vs\ xx\ <\ NN\ then\ 12\ else$
 $ibT\ ibUT\ [Config\ (if\ vs\ xx\ <\ NN\ then\ 12\ else$

γ) (*State* (*Vstore vs*) *avst hh p*)]
 $ls \cup readLocs$ (*Config pc* (*State* (*Vstore vs*)
avst hh p)) *ls*])
prefer 9 subgoal by metis by (*simp add: finalM-iff*)+

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)
 $pstate pstate ibT ibUT$
 $Config$ (*if vs xx < NN then 7 else 12*) (*State* (*Vstore*
vs) *avst hh p*)
 $ibT ibUT [] ls \cup readLocs$ (*Config pc* (*State* (*Vstore*
vs) *avst hh p*)) *ls*])
prefer 7 subgoal by metis by simp-all .

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)
 $pstate pstate ibT ibUT$
(*let l = (array-loc aa1 (nat (vs xx)) (Avstore as))*
in (Config 8 (State (Vstore (vs(vv := h l))) avst hh p)))
 $ibT ibUT [] ls \cup readLocs$ (*Config pc* (*State* (*Vstore vs*) *avst*
hh p)) *ls*])
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)
 $pstate pstate ibT ibUT$
(*Config 9 (State (Vstore vs) avst hh p)*)
 $ibT ibUT [] ls \cup readLocs$ (*Config pc* (*State* (*Vstore vs*) *avst*
hh p)) *ls*])
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)
 $pstate pstate ibT ibUT$
 $Config 10$ (*State* (*Vstore vs*) *avst hh p*)
 $ibT ibUT [] ls ls$])
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)
 $pstate pstate ibT ibUT$
(*let l = (array-loc aa2 (nat (vs vv * 512)) (Avstore as))*
in (Config 11 (State (Vstore (vs(tt := h l))) avst hh p)))
 $ibT ibUT [] ls \cup readLocs$ (*Config pc* (*State* (*Vstore vs*) *avst*
hh p)) *ls*])
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[*of cfgs Config pc* (*State* (*Vstore vs*) *avst*
hh p)

pstate pstate ibT ibUT
Config 12 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls]

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal[of cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT
Config 3 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p)) ls]

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal[of cfgs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT
Config 14 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls]

prefer 7 subgoal by metis by simp-all
by simp-all . . .

lemma *finalS-cond':pcOf cfg < 14 ⇒ cfgs = [] ⇒ ibT ≠ LNil ⇒ ibUT ≠ LNil ⇒*
 \neg *finalS (pstate, cfg, cfgs, ibT, ibUT, ls)*
using *finalS-cond* **by** (*simp add: noMisSpec-def*)

lemma *finalS-while-spec:*
whileSpeculation cfg (last cfgs) ⇒
length cfgs = Suc 0 ⇒
 \neg *finalS (pstate, cfg, cfgs, ibT, ibUT, ls)*
apply(*unfold whileSpec-defs, cases cfg*)
subgoal for *pc s* **apply**(*cases s*)
subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
apply(*elim disjE, elim conjE*) **unfolding** *finalS-defs*
subgoal apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal using *stepS-spec-resolve-iff[of cfgs pstate cfg ibT ibUT ls update pstate (pcOf cfg # map pcOf cfgs)]* **by** *fastforce*
subgoal using *spec-resolve[of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls]* **by** *fastforce .*
subgoal apply(*elim conjE, cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal using *spec-resolve[of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls]* **by** *fastforce*
subgoal using *spec-resolve[of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg [] ibT ibT ibUT ibUT ls ls]* **by** *fastforce*

lemma *finalS-while-spec-L2:*
pcOf cfg = 7 ⇒
whileSpeculation (cfgs!0) (last cfgs) ⇒

$length\ cfgs = 2 \implies$
 $\neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
apply(*unfold whileSpec-defs, cases cfg*)
subgoal for *pc s* **apply**(*cases s*)
subgoal for *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
apply(*elim disjE, elim conjE*) **unfolding** *finalS-defs*
subgoal apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal using *stepS-spec-resolve-iff*[*of cfgs pstate cfg ibT ibUT ls update pstate (pcOf cfg # map pcOf cfgs)*] **by** *fastforce*
subgoal using *spec-resolve*[*of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls*] **by** *fastforce* .
apply(*elim conjE, cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
using *spec-resolve*[*of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls*] **by** *fastforce*+ . . .

lemma *finalS-if-spec*:

$(pcOf\ (last\ cfgs) \in inThenIfBeforeOutput \wedge pcOf\ cfg = 12) \vee$
 $(pcOf\ (last\ cfgs) \in inElseIf \wedge pcOf\ cfg = 7) \implies$
 $length\ cfgs = Suc\ 0 \implies$
 $\neg finalS\ (pstate, cfg, cfgs, ibT, ibUT, ls)$
unfolding *inThenIfBeforeOutput-def inElseIf-def*
apply(*simp, cases last cfgs*)
subgoal for *pc s* **apply**(*cases s*)
subgoal for *vst avst hh p* **apply**(*cases vst, cases hh*)
subgoal for *vs h*
apply(*elim disjE, elim conjE*) **unfolding** *finalS-defs*
subgoal apply(*elim disjE*)
subgoal apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal using *spec-resolve*[*of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls*] **by** *fastforce*
apply(*rule notI,*
erule allE[*of - (pstate, cfg,*
 $[Config\ 8\ (State\ (Vstore\ (vs(vv := h\ (array-loc\ aa1\ (nat\ (vs\ xx))\ avst))\ avst\ hh\ p)],$
 $ibT, ibUT, ls \cup readLocs\ (last\ cfgs))]$]
by (*erule notE,*
rule spec-normal[*of - - - - Config 8 (State (Vstore (vs(vv := h (array-loc aa1 (nat (vs xx)) avst)) avst hh p)], auto)*]
subgoal apply(*rule notI, erule allE*[*of - (update pstate (pcOf cfg # map pcOf cfgs), cfg, [], ibT, ibUT, ls \cup readLocs (last cfgs))]*)
by(*erule notE, rule spec-resolve, auto*) .
subgoal apply(*elim conjE, elim disjE*)
subgoal apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal using *spec-resolve*[*of cfgs pstate cfg update pstate (pcOf cfg # map pcOf cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls*] **by** *fastforce* **apply**(*rule notI, erule allE*[*of -*
 $(pstate, cfg, [Config\ 3\ (State\ (Vstore\ vs)\ avst\ hh\ p)], ibT, ibUT,$
 $ls \cup readLocs\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ hh\ p))]$]

```

      by(erule notE, rule spec-normal[of - - - - -Config 3 (State (Vstore vs)
avst hh p)], auto)

      apply(cases resolve pstate (pcOf cfg # map pcOf cfgs))
      subgoal using spec-resolve[of cfgs pstate cfg update pstate (pcOf cfg # map
pcOf cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls ] by fastforce
      subgoal apply(cases mispred pstate [7,3])
      subgoal apply(rule notI, erule allE[of -
(update pstate (pcOf cfg # map pcOf cfgs),
cfg,
[Config (if vs xx ≠ 0 then 4 else 13) (State (Vstore vs) avst hh p),
Config (if vs xx ≠ 0 then 13 else 4) (State (Vstore vs) avst hh p)], ibT,
ibUT,
ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p))]))
      apply(erule notE,
rule spec-mispred[of - - - - -
Config (if vs xx ≠ 0 then 4 else 13) (State (Vstore vs) avst hh p) -
- Config (if vs xx ≠ 0 then 13 else 4) (State (Vstore vs) avst hh p) ibT
ibUT])
      by(auto simp: finalM-iff)

      apply(rule notI, erule allE[of -
(pstate, cfg, [Config (if vs xx ≠ 0 then 4 else 13) (State (Vstore vs) avst
hh p)], ibT,ibUT,
ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p))]))
      by (erule notE,
rule spec-normal[of - - - - -Config (if vs xx ≠ 0 then 4 else 13) (State
(Vstore vs) avst hh p)], auto)

      using spec-resolve[of cfgs pstate cfg update pstate (pcOf cfg # map pcOf
cfgs) cfg butlast cfgs ibT ibT ibUT ibUT ls ls ] by fastforce+
      . . . .

```

end

13.2 Proof

```

theory Fun6-secure
imports Fun6
begin

```

```

definition common :: enat ⇒ enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV ⇒
stateV ⇒ status ⇒ bool

```

```

where

```

```

common = (λw1 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* $w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$

$(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg1 < 14 \wedge pcOf\ cfg2 = pcOf\ cfg1 \wedge$
 $ibT1 \neq [] \wedge ibT2 \neq [] \wedge$
 $ibUT1 \neq [] \wedge ibUT2 \neq [] \wedge$
 $w1 = w2$

unfolding *common-def PC-def* **by** *(auto simp: image-def subset-eq)*

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 = []
apply (meson Δ0-def common-implies)
by (simp-all add: Δ0-defs, metis Nil-is-map-conv)

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)

```

$statO \wedge$
 $pcOf\ cfg3 \in afterWhile \wedge$
 $same-var-o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $noMisSpec\ cfs3$
 $)$
lemmas $\Delta1-defs = \Delta1-def\ common-def\ noMisSpec-def\ PC-def\ afterWhile-def\ same-var-o-def$
lemma $\Delta1-implies: \Delta1\ n\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \implies$
 $pcOf\ cfg3 < 14 \wedge cfs3 = [] \wedge ibT3 \neq [] \wedge$
 $pcOf\ cfg4 < 14 \wedge cfs4 = [] \wedge ibT4 \neq [] \wedge$
 $ibUT3 \neq [] \wedge ibUT4 \neq []$
unfolding $\Delta1-defs\ apply\ clarify$
by $(metis\ atLeastAtMost-iff\ eval-nat-numeral(2)\ infinity-ne-i0$
 $less-Suc-eq-le\ list.map-disc-iff\ llength-LNil\ semiring-norm(28))$

definition $\Delta1' :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV$
 $\Rightarrow stateV \Rightarrow status \Rightarrow bool$ **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 $\Delta1'-implies: \Delta1'\ num\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$

$(\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\text{statO} \implies$
 $\text{pcOf } \text{cfg3} < 14 \wedge \text{pcOf } \text{cfg4} < 14 \wedge$
 $\text{whileSpeculation } \text{cfg3} (\text{last } \text{cfigs3}) \wedge$
 $\text{whileSpeculation } \text{cfg4} (\text{last } \text{cfigs4}) \wedge$
 $\text{length } \text{cfigs3} = \text{Suc } 0 \wedge \text{length } \text{cfigs4} = \text{Suc } 0$
unfolding $\Delta 1'$ -defs by clarsimp

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

$\Delta 2 = (\lambda \text{num } w1 \ w2 \ (\text{pstate3}, \text{cfg3}, \text{cfigs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $\quad (\text{pstate4}, \text{cfg4}, \text{cfigs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 $\quad \text{statA}$
 $\quad (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $\quad (\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\quad \text{statO}.$
 $(\text{common } w1 \ w2 \ (\text{pstate3}, \text{cfg3}, \text{cfigs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $\quad (\text{pstate4}, \text{cfg4}, \text{cfigs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 $\quad \text{statA}$
 $\quad (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $\quad (\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\quad \text{statO} \wedge$

$\text{same-var-o } xx \ \text{cfg3} \ \text{cfigs3} \ \text{cfg4} \ \text{cfigs4} \wedge$
 $\text{pcOf } \text{cfg3} = \text{startOfIfThen} \wedge \text{pcOf } (\text{last } \text{cfigs3}) \in \text{inElseIf} \wedge$
 $\text{misSpecL1 } \text{cfigs3} \wedge \text{misSpecL1 } \text{cfigs4} \wedge$

$(\text{pcOf } (\text{last } \text{cfigs3}) = \text{startOfElseBranch} \longrightarrow w1 = \infty) \wedge$
 $(\text{pcOf } (\text{last } \text{cfigs3}) = 3 \longrightarrow w1 = 3) \wedge$

$(\text{pcOf } (\text{last } \text{cfigs3}) = \text{startOfWhileThen} \vee$
 $\text{pcOf } (\text{last } \text{cfigs3}) = \text{whileElse} \longrightarrow w1 = 1)$

))

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: $\Delta 2 \ \text{num } w1 \ w2 \ (\text{pstate3}, \text{cfg3}, \text{cfigs3}, \text{ibT3}, \text{ibUT3}, \text{ls3})$
 $\quad (\text{pstate4}, \text{cfg4}, \text{cfigs4}, \text{ibT4}, \text{ibUT4}, \text{ls4})$
 $\quad \text{statA}$
 $\quad (\text{cfg1}, \text{ibT1}, \text{ibUT1}, \text{ls1})$
 $\quad (\text{cfg2}, \text{ibT2}, \text{ibUT2}, \text{ls2})$
 $\quad \text{statO} \implies$
 $\text{pcOf } (\text{last } \text{cfigs3}) \in \text{inElseIf} \wedge \text{pcOf } \text{cfg3} = 7 \wedge$
 $\text{pcOf } (\text{last } \text{cfigs4}) = \text{pcOf } (\text{last } \text{cfigs3}) \wedge$

$pcOf\ cfg4 = pcOf\ cfg3 \wedge length\ cfs3 = Suc\ 0 \wedge$
 $length\ cfs4 = Suc\ 0 \wedge same-var\ xx\ (last\ cfs3)\ (last\ cfs4)$
apply(*intro conjI*)
unfolding $\Delta 2$ -*defs*
apply (*simp-all add: image-subset-iff*)
by (*metis last-in-set length-0-conv Nil-is-map-conv last-map length-map*)+

definition $\Delta 2' :: enat \Rightarrow enat \Rightarrow enat \Rightarrow stateO \Rightarrow stateO \Rightarrow status \Rightarrow stateV$
 $\Rightarrow stateV \Rightarrow status \Rightarrow bool$ **where**
 $\Delta 2' = (\lambda num\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO.$
 $(common\ w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \wedge$
 $same-var-o\ xx\ cfg3\ cfs3\ cfg4\ cfs4 \wedge$
 $pcOf\ cfg3 = startOfIfThen \wedge$
 $whileSpeculation\ (cfs3!0)\ (last\ cfs3) \wedge$
 $misSpecL2\ cfs3 \wedge misSpecL2\ cfs4 \wedge$
 $w1 = 2$
 $))$

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* $w1\ w2\ (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)$
 $(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)$
 $statA$
 $(cfg1, ibT1, ibUT1, ls1)$
 $(cfg2, ibT2, ibUT2, ls2)$
 $statO \Longrightarrow$
 $pcOf\ cfg3 = 7 \wedge pcOf\ cfg4 = 7 \wedge$
 $pcOf\ (last\ cfs3) = pcOf\ (last\ cfs4) \wedge$
 $whileSpeculation\ (cfs3!0)\ (last\ cfs3) \wedge$
 $whileSpeculation\ (cfs4!0)\ (last\ cfs4) \wedge$
 $length\ cfs3 = 2 \wedge length\ cfs4 = 2$
apply(*intro conjI*)
unfolding $\Delta 2'$ -*defs* **apply** (*simp add: lessI, clarify*)
apply *linarith+* **apply** *simp-all*
apply (*metis last-map length-0-conv*)
by (*metis list.inject map-L2*)


```

  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 $\Delta e\text{-defs} = \Delta e\text{-def}$ *common-def* *endPC-def*

```

lemma init: initCond  $\Delta 0$ 
unfolding initCond-def apply safe
  subgoal for pstate3 cfg3 cfs3 ibT3 ibUT3 ls3
    pstate4 cfg4 cfs4 ibT4 ibUT4 ls4
  unfolding istateO.simps apply clarsimp
apply(cases getAvstore (stateOf cfg3), cases getAvstore (stateOf cfg4))
unfolding  $\Delta 0\text{-defs}$ 
unfolding array-base-def by auto .

```

lemma *step0: unwindIntoCond* $\Delta 0$ (*oor* $\Delta 0$ $\Delta 1$)

```

proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 0$ :  $\Delta 0$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

```

```

  obtain pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

```

```

  obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

```

```

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)

```

```

obtain  $hh_4$  where  $h_4: h_4 = \text{Heap } hh_4$  by(cases  $h_4$ , auto)
note  $hh = h_3 h_4$ 

have  $f_1: \neg \text{finalN } ss_1$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-by(frule  $\Delta 0$ -implies, simp)

have  $f_2: \neg \text{finalN } ss_2$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-by(frule  $\Delta 0$ -implies, simp)

have  $f_3: \neg \text{finalS } ss_3$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-apply(frule  $\Delta 0$ -implies, unfold  $\Delta 0$ -defs)
  using finalS-cond' by simp

have  $f_4: \neg \text{finalS } ss_4$ 
  using  $\Delta 0$  unfolding  $ss$ 
  apply-apply(frule  $\Delta 0$ -implies, unfold  $\Delta 0$ -defs)
  using finalS-cond' by simp

note  $\text{finals} = f_1 f_2 f_3 f_4$ 
show  $\text{finalS } ss_3 = \text{finalS } ss_4 \wedge \text{finalN } ss_1 = \text{finalS } ss_3 \wedge \text{finalN } ss_2 = \text{finalS } ss_4$ 
  using  $\text{finals}$  by auto

then show  $\text{isIntO } ss_3 = \text{isIntO } ss_4$  by simp

show  $\text{react } (\text{oor } \Delta 0 \Delta 1) w_1 w_2 ss_3 ss_4 \text{ statA } ss_1 ss_2 \text{ statO}$ 
unfolding react-def proof(intro conjI)

  show  $\text{match1 } (\text{oor } \Delta 0 \Delta 1) w_1 w_2 ss_3 ss_4 \text{ statA } ss_1 ss_2 \text{ statO}$ 
  unfolding match1-def by (simp add: finalS-def final-def)
  show  $\text{match2 } (\text{oor } \Delta 0 \Delta 1) w_1 w_2 ss_3 ss_4 \text{ statA } ss_1 ss_2 \text{ statO}$ 
  unfolding match2-def by (simp add: finalS-def final-def)
  show  $\text{match12 } (\text{oor } \Delta 0 \Delta 1) w_1 w_2 ss_3 ss_4 \text{ statA } ss_1 ss_2 \text{ statO}$ 

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix  $ss_3' ss_4' \text{ statA}'$ 
  assume  $\text{statA}': \text{statA}' = \text{sstatA}' \text{ statA } ss_3 ss_4$ 
  and  $v: \text{validTransO } (ss_3, ss_3') \text{ validTransO } (ss_4, ss_4')$ 
  and  $sa: \text{Opt.eqAct } ss_3 ss_4$ 
  note  $v_3 = v(1)$  note  $v_4 = v(2)$ 

  obtain  $pstate_3' cfg_3' cfs_3' ibT_3' ibUT_3' ls_3'$  where  $ss_3': ss_3' = (pstate_3',$ 
 $cfg_3', cfs_3', ibT_3', ibUT_3', ls_3')$ 
  by (cases  $ss_3'$ , auto)
  obtain  $pstate_4' cfg_4' cfs_4' ibT_4' ibUT_4' ls_4'$  where  $ss_4': ss_4' = (pstate_4',$ 
 $cfg_4', cfs_4', ibT_4', ibUT_4', ls_4')$ 

```

```

by (cases ss4', auto)
note ss = ss ss3' ss4'

obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

show eqSec ss1 ss3
  using v sa Δ0 finals unfolding ss
  by (simp add: Δ0-defs eqSec-def)

show eqSec ss2 ss4
  using v sa Δ0 finals unfolding ss
  by (simp add: Δ0-defs eqSec-def, metis map-is-Nil-conv)

show Van.eqAct ss1 ss2
  using v sa Δ0 unfolding ss
  apply-apply(frule Δ0-implies)
  unfolding Opt.eqAct-def
    Van.eqAct-def
  by(simp-all add: Δ0-defs, linarith)

show match12-12 (oor Δ0 Δ1) ∞ ∞ ss3' ss4' statA' ss1 ss2 statO
  unfolding match12-12-def
  proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
  using v sa Δ0 sstat unfolding ss cfg statA' apply simp
  apply(simp add: Δ0-defs sstatO'-def sstatA'-def finalS-def final-def)
  using cases-14[of pc3] apply(elim disjE)
  apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
  apply(cases statO, simp-all) apply (cases statA, simp-all)
  by (smt (z3) status.distinct status.exhaust newStat.simps)+
  } note stat = this

  show oor Δ0 Δ1 ∞ ∞ ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO'
statO ss1 ss2)

```

```

using  $v3$ [unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$ 
  by (simp add:  $\Delta 0$ -defs numeral-1-eq-Suc-0, linarith)
next
  case spec-normal
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case spec-mispred
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case spec-Fence
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case spec-resolve
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case spec-resolveI
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case spec-resolveO
  then show  $?thesis$  using  $sa \Delta 0$  stat unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
next
  case nonspec-normal note  $nn3 = nonspec-normal$ 
  show  $?thesis$ 
  using  $v3$ [unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-mispred
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
    next
    case spec-normal
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
    next
    case spec-mispred
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
    next
    case spec-Fence
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
    next
    case spec-resolve
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)
    next
    case spec-resolveI
    then show  $?thesis$  using  $sa \Delta 0$  stat  $nn3$  unfolding  $ss$  by (simp add:  $\Delta 0$ -defs)

```

```

      next
      case spec-resolveO
      then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
      next
      case nonspec-normal note nn4 = nonspec-normal
      show ?thesis using sa  $\Delta 0$  stat v3 v4 nn3 nn4 unfolding ss cfg apply
clarsimp
      apply(unfold  $\Delta 0$ -defs, clarsimp, elim disjE)
      subgoal by(rule oorI1, auto simp add:  $\Delta 0$ -defs)
      subgoal by (rule oorI1, simp add:  $\Delta 0$ -defs)
      subgoal by (rule oorI2, simp add:  $\Delta 1$ -defs) .
      qed
      qed
      qed
      qed
      qed
      qed

```

lemma *step1: unwindIntoCond* $\Delta 1$ (oor5 $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe)

proof(rule *unwindIntoCond-simpleI*)

fix $n w1 w2 ss3 ss4 statA ss1 ss2 statO$

assume r : reachO $ss3$ reachO $ss4$ reachV $ss1$ reachV $ss2$

and $\Delta 1$: $\Delta 1 n w1 w2 ss3 ss4 statA ss1 ss2 statO$

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by (cases $ss3$, auto)

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by (cases $ss4$, auto)

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (cases $ss1$, auto)

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (cases $ss2$, auto)

note $ss = ss3 ss4 ss1 ss2$

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$cfg3$: $cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)$

by (cases $cfg3$) (metis state.collapse vstore.collapse)

obtain $pc4$ $vs4$ $avst4$ $h4$ $p4$ **where**

$cfg4$: $cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)$

by (cases $cfg4$) (metis state.collapse vstore.collapse)

note $cfg = cfg3 cfg4$

obtain $hh3$ **where** $h3$: $h3 = Heap hh3$ **by**(cases $h3$, auto)

obtain $hh4$ **where** $h4$: $h4 = Heap hh4$ **by**(cases $h4$, auto)

```

note  $hh = h3\ h4$ 

have  $f1:\neg finalN\ ss1$ 
  using  $\Delta1$  unfolding  $ss\ \Delta1-def$  apply clarify
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by metis

have  $f2:\neg finalN\ ss2$ 
  using  $\Delta1$  unfolding  $ss\ \Delta1-def$  apply clarify
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by metis

have  $f3:\neg finalS\ ss3$ 
  using  $\Delta1$  unfolding  $ss$ 
  apply–apply(frule \Delta1-implies)
  by (simp add: finalS-cond')

have  $f4:\neg finalS\ ss4$ 
  using  $\Delta1$  unfolding  $ss$ 
  apply–apply(frule \Delta1-implies)
  by (simp add: finalS-cond')

note  $finals = f1\ f2\ f3\ f4$ 
show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
  using  $finals$  by auto

then show  $isIntO\ ss3 = isIntO\ ss4$  by simp

show  $react\ (oor5\ \Delta1\ \Delta1'\ \Delta2\ \Delta3\ \Delta e)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding react-def proof(intro conjI)

  show  $match1\ (oor5\ \Delta1\ \Delta1'\ \Delta2\ \Delta3\ \Delta e)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding match1-def by (simp add: finalS-def final-def)
  show  $match2\ (oor5\ \Delta1\ \Delta1'\ \Delta2\ \Delta3\ \Delta e)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding match2-def by (simp add: finalS-def final-def)
  show  $match12\ (oor5\ \Delta1\ \Delta1'\ \Delta2\ \Delta3\ \Delta e)\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 

proof(rule match12-simpleI,rule disjI2, intro conjI)
  fix  $ss3'\ ss4'\ statA'$ 
  assume  $statA': statA' = sstatA'\ statA\ ss3\ ss4$ 
  and  $v:$  validTransO ( $ss3, ss3'$ ) validTransO ( $ss4, ss4'$ )
  and  $sa:$  Opt.eqAct  $ss3\ ss4$ 
  note  $v3 = v(1)$  note  $v4 = v(2)$ 

  obtain  $pstate3'\ cfg3'\ cfs3'\ ibT3'\ ibUT3'\ ls3'$  where  $ss3': ss3' = (pstate3',$ 
 $cfg3', cfs3', ibT3', ibUT3', ls3')$ 
  by (cases ss3', auto)

```

```

obtain  $pstate4'$   $cfg4'$   $cfgs4'$   $ibT4'$   $ibUT4'$   $ls4'$  where  $ss4'$ :  $ss4' = (pstate4',$ 
 $cfg4', cfgs4', ibT4', ibUT4', ls4')$ 
by ( $cases\ ss4'$ ,  $auto$ )
note  $ss = ss\ ss3'\ ss4'$ 

show  $eqSec\ ss1\ ss3$ 
using  $v\ sa\ \Delta1\ finals\ unfolding\ ss\ by\ (simp\ add:\ \Delta1-defs\ eqSec-def)$ 

show  $eqSec\ ss2\ ss4$ 
using  $v\ sa\ \Delta1\ finals\ unfolding\ ss$ 
by ( $simp\ add:\ \Delta1-defs\ eqSec-def$ ,  $metis\ map-is-Nil-conv$ )

show  $Van.eqAct\ ss1\ ss2$ 
using  $v\ sa\ \Delta1\ unfolding\ ss\ apply-apply(frule\ \Delta1-implies)$ 
unfolding  $Opt.eqAct-def\ Van.eqAct-def$ 
apply( $simp-all\ add:\ \Delta1-defs$ )
by ( $metis\ f3\ getActO-pcOf\ numeral-eq-iff\ numeral-less-iff\ semiring-norm(77,78,81,89)$ 
 $ss3$ )

show  $match12-12\ (oor5\ \Delta1\ \Delta1'\ \Delta2\ \Delta3\ \Delta e)\ \infty\ \infty\ ss3'\ ss4'\ statA'\ ss1\ ss2$ 
 $statO$ 
unfolding  $match12-12-def$ 
proof( $rule\ exI[of\ -\ nextN\ ss1]$ ,  $rule\ exI[of\ -\ nextN\ ss2]$ ,  $unfold\ Let-def$ ,  $intro$ 
 $conjI\ impI$ )
show  $validTransV\ (ss1,\ nextN\ ss1)$ 
by ( $simp\ add:\ f1\ nextN-stepN$ )

show  $validTransV\ (ss2,\ nextN\ ss2)$ 
by ( $simp\ add:\ f2\ nextN-stepN$ )

{assume  $sstat:\ statA' = Diff$ 
show  $sstatO'\ statO\ ss1\ ss2 = Diff$ 
using  $v\ sa\ \Delta1\ sstat\ finals\ unfolding\ ss\ cfg\ statA'$ 
apply-apply( $frule\ \Delta1-implies$ )
apply( $simp\ add:\ \Delta1-defs\ sstatO'-def\ sstatA'-def\ newStat-EqI$ )
using  $cases-14[of\ pc3]\ apply(elim\ disjE,\ simp-all)$ 
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all$ )
by( $cases\ statA,\ simp-all\ add:\ newStat-EqI$ )
subgoal apply( $cases\ statO,\ simp-all,\ cases\ statA$ )

```

```

    by (simp-all add: newStat-EqI)
  subgoal apply(cases statO, simp-all)
    by(cases statA, simp-all add: newStat-EqI)
  subgoal apply(cases statO, simp-all, cases statA)
    by (simp-all add: newStat-EqI split: if-splits)
  subgoal apply(cases statO, simp-all, cases statA)
    by (simp-all add: newStat-EqI split: if-splits)
  subgoal apply(cases statO, simp-all, cases statA)
    by (simp-all add: newStat-EqI split: if-splits) .
} note stat = this

show oor5  $\Delta 1$   $\Delta 1'$   $\Delta 2$   $\Delta 3$   $\Delta e$   $\infty$   $\infty$   $\infty$   $ss3'$   $ss4'$   $statA'$  ( $nextN$   $ss1$ ) ( $nextN$ 
 $ss2$ ) ( $sstatO'$   $statO$   $ss1$   $ss2$ )

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case spec-normal
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-mispred
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-Fence
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-resolve
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-resolveI
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case spec-resolveO
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

    case nonspec-mispred
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-normal
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)

```

```

      next
      case spec-mispred
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
      next
      case spec-Fence
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
      next
      case spec-resolve
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
      next
      case spec-resolveI
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
      next
      case spec-resolveO
      then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
      next
      case nonspec-normal note nn4 = nonspec-normal
      then show ?thesis using sa  $\Delta 1$  stat v3 v4 nn3 nn4 f4 unfolding ss cfg
Opt.eqAct-def
      apply clarsimp using cases-14[of pc3] apply(elim disjE)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal by (simp add:  $\Delta 1$ -defs)
      subgoal using xx-0-cases[of vs3] apply(elim disjE)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs) .
      subgoal apply(rule oor5I1) by (auto simp add:  $\Delta 1$ -defs)
      subgoal apply(rule oor5I1) by (auto simp add:  $\Delta 1$ -defs)
      subgoal using xx-NN-cases[of vs3] apply(elim disjE)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs) .
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs hh)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs hh)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs hh)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      subgoal by(rule oor5I1, auto simp add:  $\Delta 1$ -defs)
      by(rule oor5I5, simp-all add:  $\Delta 1$ -defs  $\Delta e$ -defs)
      qed
      next
      case nonspec-mispred note nm3 = nonspec-mispred
      show ?thesis using v4 [unfolded ss, simplified] proof(cases rule: stepS-cases)

      case nonspec-normal

```

```

      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-normal
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-mispred
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-Fence
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-resolve
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-resolveI
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case spec-resolveO
      then show ?thesis using sa  $\Delta 1$  stat nm3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
    next
      case nonspec-mispred note nm4 = nonspec-mispred
      then show ?thesis using sa  $\Delta 1$  stat v3 v4 nm3 nm4 unfolding ss cfg
apply clarsimp
      using cases-14[of pc3] apply(elim disjE)
      prefer 4 subgoal using xx-0-cases[of vs3] apply(elim disjE)
      subgoal by(rule oor5I2, auto simp add:  $\Delta 1$ -defs  $\Delta 1'$ -defs)
      subgoal by(rule oor5I2, auto simp add:  $\Delta 1$ -defs  $\Delta 1'$ -defs) .
      prefer 6 subgoal using xx-NN-cases[of vs3] apply(elim disjE)
      subgoal apply(rule oor5I3) by (auto simp add:  $\Delta 1$ -defs  $\Delta 2$ -defs)
      subgoal apply(rule oor5I4) by (auto simp add:  $\Delta 1$ -defs  $\Delta 3$ -defs) .
      by (simp-all add:  $\Delta 1$ -defs)
    qed
  qed
qed
qed
qed
qed

```

lemma step2: unwindIntoCond $\Delta 2$ (oor3 $\Delta 2$ $\Delta 2'$ $\Delta 1$)
proof(rule unwindIntoCond-simpleI)

```

fix  $n\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
assume  $r: reachO\ ss3\ reachO\ ss4\ reachV\ ss1\ reachV\ ss2$ 
and  $\Delta2: \Delta2\ n\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 

obtain  $pstate3\ cfg3\ cfgs3\ ibT3\ ibUT3\ ls3$  where  $ss3: ss3 = (pstate3, cfg3, cfgs3,$ 
 $ibT3, ibUT3, ls3)$ 
by (cases  $ss3$ , auto)
obtain  $pstate4\ cfg4\ cfgs4\ ibT4\ ibUT4\ ls4$  where  $ss4: ss4 = (pstate4, cfg4, cfgs4,$ 
 $ibT4, ibUT4, ls4)$ 
by (cases  $ss4$ , auto)
obtain  $cfg1\ ibT1\ ibUT1\ ls1$  where  $ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)$ 
by (cases  $ss1$ , auto)
obtain  $cfg2\ ibT2\ ibUT2\ ls2$  where  $ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)$ 
by (cases  $ss2$ , auto)
note  $ss = ss3\ ss4\ ss1\ ss2$ 

obtain  $pc3\ vs3\ avst3\ h3\ p3$  where
 $lcfgs3: last\ cfigs3 = Config\ pc3\ (State\ (Vstore\ vs3)\ avst3\ h3\ p3)$ 
by (cases  $last\ cfigs3$ ) (metis  $state.collapse\ vstore.collapse$ )
obtain  $pc4\ vs4\ avst4\ h4\ p4$  where
 $lcfgs4: last\ cfigs4 = Config\ pc4\ (State\ (Vstore\ vs4)\ avst4\ h4\ p4)$ 
by (cases  $last\ cfigs4$ ) (metis  $state.collapse\ vstore.collapse$ )
note  $lcfigs = lcfgs3\ lcfgs4$ 

have  $f1: \neg finalN\ ss1$ 
using  $\Delta2$  unfolding  $ss\ \Delta2-def$ 
apply clarsimp
by(frule common-implies, simp)

have  $f2: \neg finalN\ ss2$ 
using  $\Delta2$  unfolding  $ss\ \Delta2-def$ 
apply clarsimp
by(frule common-implies, simp)

have  $f3: \neg finalS\ ss3$ 
using  $\Delta2$  unfolding  $ss$ 
apply–apply(frule  $\Delta2-implies$ )
by (simp add: finalS-if-spec)

have  $f4: \neg finalS\ ss4$ 
using  $\Delta2$  unfolding  $ss$ 
apply–apply(frule  $\Delta2-implies$ )
by (simp add: finalS-if-spec)

note  $finals = f1\ f2\ f3\ f4$ 
show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
using  $finals$  by auto

```

then show $isIntO\ ss3 = isIntO\ ss4$ **by** *simp*

then have $lpc3:pcOf\ (last\ cfigs3) = 12 \vee$
 $pcOf\ (last\ cfigs3) = 3 \vee$
 $pcOf\ (last\ cfigs3) = 4 \vee$
 $pcOf\ (last\ cfigs3) = 13$
using $\Delta 2$ **unfolding** *ss* $\Delta 2$ -*defs* **by** *simp*

have $sec3[simp]: \neg isSecO\ ss3$
using $\Delta 2$ *finals* **unfolding** *ss* *isSecO-def*
by (*simp* *add*: $\Delta 2$ -*defs*, *metis* *list.size(3)* *n-not-Suc-n*)

have $sec4[simp]: \neg isSecO\ ss4$
using $\Delta 2$ **unfolding** *ss*
by (*simp* *add*: $\Delta 2$ -*defs*, *metis* *list.size(3)* *n-not-Suc-n*)

have $stat[simp]: \bigwedge ss3'\ ss4'\ statA'. statA' = sstatA'\ statA\ ss3\ ss4 \implies$
 $validTransO\ (ss3, s3') \implies validTransO\ (ss4, s4') \implies$
 $(statA = statA' \vee statO = Diff)$

subgoal for $ss3'\ ss4'$
apply (*cases* *ss3*, *cases* *ss4*, *cases* *ss1*, *cases* *ss2*)
apply (*cases* *ss3'*, *cases* *ss4'*, *clarsimp*)
using $\Delta 2$ *finals* **unfolding** *ss* **apply** *clarsimp*
apply (*simp-all* *add*: $\Delta 2$ -*defs* *sstatA'-def*)
apply (*cases* *statO*, *simp-all*) **by** (*cases* *statA*, *simp-all* *add*: *newStat-EqI*) .

have $xx:vs3\ xx = vs4\ xx$ **using** $\Delta 2$ *lcfigs* **unfolding** *ss* $\Delta 2$ -*defs* **apply** *clarsimp*
by (*metis* *cfigs-Suc-zero* *config.sel(2)* *list.set-intros(1)* *state.sel(1)* *vstore.sel*)

have $oor3$ -*rule*: $\bigwedge ss3'\ ss4'. ss3 \rightarrow_S ss3' \implies ss4 \rightarrow_S ss4' \implies$
 $(pcOf\ (last\ cfigs3) = 12 \longrightarrow oor3\ \Delta 2\ \Delta 2'\ \Delta 1 \infty 3\ 3\ ss3'\ ss4'$
 $(sstatA'\ statA\ ss3\ ss4)\ ss1\ ss2\ statO)$
 $\wedge (pcOf\ (last\ cfigs3) = 3 \wedge mispred\ pstate4\ [7, 3] \longrightarrow oor3\ \Delta 2\ \Delta 2'$
 $\Delta 1 \infty 2\ 2\ ss3'\ ss4'\ (sstatA'\ statA\ ss3\ ss4)\ ss1\ ss2\ statO)$
 $\wedge (pcOf\ (last\ cfigs3) = 3 \wedge \neg mispred\ pstate4\ [7, 3] \longrightarrow oor3\ \Delta 2$
 $\Delta 2'\ \Delta 1 \infty 1\ 1\ ss3'\ ss4'\ (sstatA'\ statA\ ss3\ ss4)\ ss1\ ss2\ statO)$
 $\wedge ((pcOf\ (last\ cfigs3) = 4 \vee pcOf\ (last\ cfigs3) = 13) \longrightarrow oor3\ \Delta 2$
 $\Delta 2'\ \Delta 1 \infty 0\ 0\ ss3'\ ss4'\ (sstatA'\ statA\ ss3\ ss4)\ ss1\ ss2\ statO) \implies$
 $\exists w1' < w1. \exists w2' < w2. oor3\ \Delta 2\ \Delta 2'\ \Delta 1 \infty w1'\ w2'\ ss3'\ ss4'$
 $(sstatA'\ statA\ ss3\ ss4)\ ss1\ ss2\ statO$
subgoal for $ss3'\ ss4'$ **apply** (*cases* *ss3'*, *cases* *ss4'*)
subgoal for $pstate3'\ cfig3'\ cfigs3'\ ibT3'\ ibUT3'\ ls3'$
 $pstate4'\ cfig4'\ cfigs4'\ ibT4'\ ibUT4'\ ls4'$
subgoal premises *p* **using** *lpc3* **apply**-**apply** (*erule* *disjE*)
subgoal apply (*intro* *exI*[*of* - 3], *intro* *conjI*)
subgoal using $\Delta 2$ **unfolding** *ss* $\Delta 2$ -*defs* **apply** *clarify*
by (*metis* *enat-ord-simps(4)* *numeral-ne-infinity*)

```

apply(intro exI[of - 3], rule conjI)
subgoal using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  by (metis enat-ord-simps(4) numeral-ne-infinity)
using p by (simp add: p)
apply(erule disjE)
subgoal apply(cases mispred pstate4 [7, 3])
  subgoal apply(intro exI[of - 2], intro conjI)
    using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
      apply (metis enat-ord-number(2) eval-nat-numeral(3) lessI)
      apply(intro exI[of - 2], rule conjI)
      using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
      apply (metis enat-ord-number(2) eval-nat-numeral(3) lessI)
      using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify
  subgoal apply(intro exI[of - 1], intro conjI)
using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis one-less-numeral-iff semiring-norm(77))
apply(intro exI[of - 1], rule conjI)
using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis one-less-numeral-iff semiring-norm(77))
using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify .
subgoal apply(intro exI[of - 0], intro conjI)
  using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis less-numeral-extra(1))
  apply(intro exI[of - 0], rule conjI)
  using  $\Delta 2$  unfolding ss  $\Delta 2$ -defs apply clarify
  apply (metis less-numeral-extra(1))
  using  $\Delta 2$  p unfolding ss  $\Delta 2$ -defs by clarify . . . .

```

```

have pstate3:pstate3 = pstate4 using  $\Delta 2$ [unfolded ss  $\Delta 2$ -defs] by fast
have pcc:pcOf (last cfs3) = pcOf (last cfs4) using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded
ss]] by auto

```

```

show react (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

```

```

show match1 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor3  $\Delta 2$   $\Delta 2'$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-simpleI, simp-all, rule disjI1)
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
  subgoal for pstate3' cfs3' cfs3' ibT3' ibUT3' ls3'
    pstate4' cfs4' cfs4' ibT4' ibUT4' ls4'
  apply-apply(rule oor3-rule, assumption+, intro conjI impI)

```

```

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal

```

```

then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case spec-mispred
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-Fence
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-resolveI
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-resolveO
  then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) cfgs4  $\neq$  []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case spec-normal
    then show ?thesis using r4 by auto
  next
    case spec-mispred
    then show ?thesis using r4 by auto
  next
    case spec-Fence
    then show ?thesis using r4 by auto
  next
    case spec-resolveO
    then show ?thesis using r4 prem pcc by auto
  next
    case spec-resolveI
    then show ?thesis using r4 prem pcc by auto
  next
    case spec-resolve note sr4 = spec-resolve

```

```

    show ?thesis using stat  $\Delta 2$  prem sr3 sr4
    unfolding ss lcfgs apply-
    apply(frul  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
    apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
    by (smt(verit) prem(1) prem(2) ss)
qed
next
  case spec-normal note sn3 = spec-normal
  then have r4:¬resolve pstate4 (pcOf cfg4 # map pcOf cfigs4) cfigs4 ≠ []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case spec-Fence
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
  next
  case spec-resolve
  then show ?thesis using r4 by auto
  next
  case spec-resolveI
  then show ?thesis using sn3 r4 pcc by auto
  next
  case spec-resolveO
  then show ?thesis using sn3 r4 pcc by auto
  next
  case spec-mispred
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
  next
  case spec-normal note sn4 = spec-normal
  have pc4:pc4 = 12 using  $\Delta 2$  prem lcfgs unfolding ss  $\Delta 2$ -defs by auto
  show ?thesis
  using  $\Delta 2$  prem sn3 sn4 finals stat unfolding ss prem(4,5) lcfgs
  apply-apply(frul  $\Delta 2$ -implies, unfold  $\Delta 2$ -defs) apply clarsimp
  apply(rule oor3I1) apply(simp-all add:  $\Delta 2$ -defs pc4)
  using final-def config.sel(2) last-in-set
  lcfgs state.sel(1,2) vstore.sel xx
  by (metis (mono-tags, lifting))
qed
qed

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)

```

```

      case nonspec-normal
      then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

    next
      case spec-Fence
      then show ?thesis using stat  $\Delta 2$  prem(6) unfolding ss by (auto simp
add:  $\Delta 2$ -defs)
    next
      case spec-normal
      then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (simp add:  $\Delta 2$ -defs,
metis cfigs-map)
    next
      case spec-resolveI
      then show ?thesis using prem(6) by fastforce
    next
      case spec-resolveO
      then show ?thesis using prem(6) by fastforce
    next
      case spec-resolve note sr3 = spec-resolve
      then have r4: resolve pstate4 (pcOf cfg4 # map pcOf cfigs4) cfigs4  $\neq$  []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
      show ?thesis using prem(2)[unfolded ss prem(5)] proof (cases rule:
stepS-cases)
        case nonspec-normal
        then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
      next
        case nonspec-mispred
        then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
      next
        case spec-normal
        then show ?thesis using r4 by auto
      next
        case spec-mispred
        then show ?thesis using r4 by auto
      next
        case spec-Fence
        then show ?thesis using r4 by auto
      next
        case spec-resolveO
        then show ?thesis using r4 prem pcc by auto
      next
        case spec-resolveI
        then show ?thesis using r4 prem pcc by auto

```

```

next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply–
  apply(frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (smt(verit) prem(1) prem(2) ss)
qed
next
case spec-mispred note sm3 = spec-mispred
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case nonspec-mispred
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolve
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolveI
  then show ?thesis using sm3  $\Delta 2$  prem pcc by (simp add:  $\Delta 2$ -defs)
next
  case spec-resolveO
  then show ?thesis using sm3  $\Delta 2$  prem pcc by (simp add:  $\Delta 2$ -defs)
next
  case spec-Fence
  then show ?thesis using sm3  $\Delta 2$  unfolding ss apply–apply(frule
 $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs)
next
  case spec-normal
  then show ?thesis using sm3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
  case spec-mispred note sm4 = spec-mispred
  have pc:pc4 = 3
  using prem(6) lcfgs  $\Delta 2$  unfolding ss apply–apply(frule  $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs )
  show ?thesis apply(rule oor3I2)
  unfolding ss  $\Delta 2'$ -def using xx-0-cases[of vs3] apply(elim disjE)
  subgoal using  $\Delta 2$  lcfgs prem pc sm3 sm4 xx finals stat unfolding ss
  apply– apply(simp add:  $\Delta 2$ -defs  $\Delta 2'$ -defs, clarify)
  apply(intro conjI)
  subgoal by (metis config.sel(2) last-in-set state.sel(1,2) vstore.sel
final-def)
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))
  subgoal by (metis config.sel(2) last-in-set state.sel(2))

```

```

      subgoal by (smt(verit) prem(1) prem(2) ss)
      subgoal by (metis config.sel(2) last-in-set state.sel(1) vstore.sel) .
    subgoal using  $\Delta 2$  lcfgs prem pc sm3 sm4 xx finals stat unfolding ss
      apply- apply(simp add:  $\Delta 2$ -defs  $\Delta 2'$ -defs, clarify)
      apply(intro conjI)
      subgoal by (metis config.sel(2) last-in-set state.sel(1,2) vstore.sel
final-def)
      subgoal by (metis config.sel(2) last-in-set state.sel(2))
      subgoal by (metis config.sel(2) last-in-set state.sel(2))
      subgoal by (metis config.sel(2) last-in-set state.sel(2))
      subgoal by (smt(verit) prem(1) prem(2) ss)
      subgoal by (metis config.sel(2) last-in-set state.sel(1) vstore.sel) . .
    qed
  qed

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

  next
  case spec-Fence
  then show ?thesis using pcc prem(6) by simp
  next
  case spec-resolveI
  then show ?thesis using pcc prem(6) by simp
  next
  case spec-resolveO
  then show ?thesis using pcc prem(6) by simp
  next
  case spec-mispred
  have  $\neg$ mispred pstate3 (pcOf cfg3 # map pcOf cfgs3) using prem(6)
pstate3  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by simp
  then show ?thesis using spec-mispred by auto
  next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) cfgs4  $\neq$  []
using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] unfolding pstate3 by auto
  show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule:
stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
  case nonspec-mispred

```

```

    then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
  next
    case spec-normal
    then show ?thesis using r4 by auto
  next
    case spec-mispred
    then show ?thesis using r4 by auto
  next
    case spec-Fence
    then show ?thesis using r4 by auto
  next
    case spec-resolveO
    then show ?thesis using r4 prem pcc by auto
  next
    case spec-resolveI
    then show ?thesis using r4 prem pcc by auto
  next
    case spec-resolve note sr4 = spec-resolve
    show ?thesis using stat  $\Delta 2$  prem sr3 sr4
    unfolding ss lcfs apply-
    apply(frul  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
    apply(rule oor3I3, simp add:  $\Delta 1$ -defs)
    by (smt(verit) prem(1) prem(2) ss)
  qed
next
case spec-normal note sn3 = spec-normal
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
case nonspec-mispred
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
next
case spec-Fence
then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
case spec-resolve
then show ?thesis using sn3 pstate3  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by
simp
next
case spec-resolveI
then show ?thesis using sn3 pstate3  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by
simp
next
case spec-resolveO
then show ?thesis using sn3 pstate3  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by
simp

```

```

next
  case spec-mispred
  then show ?thesis using sn3  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs,
metis last-map)
next
  case spec-normal note sn4 = spec-normal
  show ?thesis
  using  $\Delta 2$  lcfs prem sn3 sn4 finals unfolding ss
  apply-apply(frul  $\Delta 2$ -implies, unfold  $\Delta 2$ -defs) apply clarsimp
  apply(rule oor3I1)
  using xx-0-cases[of vs3] apply(elim disjE)
  subgoal apply(simp-all add:  $\Delta 2$ -defs, clarify)
  using config.sel(2) last-in-set stat state.sel(1,2) vstore.sel
  by (smt (verit, ccfv-SIG) Opt.final-def config.sel(1) eval-nat-numeral(3)
f3 f4 is-Output-1 le-imp-less-Suc le-refl nat-less-le ss)
  subgoal apply(simp-all add:  $\Delta 2$ -defs, clarify)
  using config.sel(2) last-in-set stat state.sel(1,2) vstore.sel
  apply(intro conjI, unfold config.sel(1))
  subgoal by simp
  subgoal by simp
  subgoal by (metis array-baseSimp)
  subgoal by (metis array-baseSimp)
  subgoal by (metis array-baseSimp)
  subgoal by (metis array-baseSimp)
  subgoal by (smt (verit) Opt.final-def ss)
  apply (smt (verit) cfs-Suc-zero lcfs list.set-intros(1))
  apply (smt (verit) cfs-Suc-zero lcfs list.set-intros(1))
  apply presburger
  by linarith+ .
qed
qed

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case spec-Fence
  then show ?thesis using stat  $\Delta 2$  prem unfolding ss by (auto simp add:
 $\Delta 2$ -defs)
  next
  case spec-mispred
  then show ?thesis using prem(6) unfolding ss by (auto simp add:  $\Delta 2$ -defs)

```

```

next
  case spec-normal
then show ?thesis using prem(6) unfolding ss by (auto simp add:  $\Delta 2$ -defs)

next
  case spec-resolveI note sr3 = spec-resolveI
show ?thesis using prem(2)[unfolded ss prem(5)] proof (cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case spec-normal
  then show ?thesis using prem(6) pcc by auto
  next
  case spec-mispred
  then show ?thesis using prem(6) pcc by auto
  next
  case spec-Fence
  then show ?thesis using prem(6) pcc by auto
  next
  case spec-resolveO
  then show ?thesis using prem(6) pcc sr3 by auto
  next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply-
  apply (frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply (rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (metis prem(1) prem(2) ss)
  next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using stat  $\Delta 2$  prem sr3 sr4
  unfolding ss lcfgs apply-
  apply (frule  $\Delta 2$ -implies) apply (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs)
  apply (rule oor3I3, simp add:  $\Delta 1$ -defs)
  by (metis prem(1) prem(2) ss)
qed
next
  case spec-resolveO note sr3 = spec-resolveO
show ?thesis using prem(2)[unfolded ss prem(5)] proof (cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 2$  sr3 unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
  case spec-normal

```

```

    then show ?thesis using prem(6) pcc by auto
  next
    case spec-mispred
    then show ?thesis using prem(6) pcc by auto
  next
    case spec-Fence
    then show ?thesis using prem(6) pcc by auto
  next
    case spec-resolveI
    then show ?thesis using prem(6) pcc sr3 by auto
  next
    case spec-resolveO note sr4 = spec-resolveO
    show ?thesis using stat Δ2 prem sr3 sr4
    unfolding ss lcfgs apply-
    apply(frul Δ2-implies) apply (simp add: Δ2-defs Δ1-defs)
    apply(rule oor3I3, simp add: Δ1-defs)
    by (metis prem(1) prem(2) ss)
  next
    case spec-resolve note sr4 = spec-resolve
    show ?thesis using stat Δ2 prem sr3 sr4
    unfolding ss lcfgs apply-
    apply(frul Δ2-implies) apply (simp add: Δ2-defs Δ1-defs)
    apply(rule oor3I3, simp add: Δ1-defs)
    by (metis prem(1) prem(2) ss)
qed
next
  case spec-resolve note sr3 = spec-resolve
  show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using stat Δ2 sr3 unfolding ss by (simp add: Δ2-defs)
  next
    case nonspec-mispred
    then show ?thesis using stat Δ2 sr3 unfolding ss by (simp add: Δ2-defs)
  next
    case spec-normal
    then show ?thesis using prem(6) pcc sr3 by auto
  next
    case spec-mispred
    then show ?thesis using prem(6) pcc sr3 by auto
  next
    case spec-Fence
    then show ?thesis using prem(6) pcc sr3 by auto
  next
    case spec-resolveI note sr4 = spec-resolveI
    show ?thesis using stat Δ2 prem sr3 sr4
    unfolding ss lcfgs apply-
    apply(frul Δ2-implies) apply (simp add: Δ2-defs Δ1-defs)
    apply(rule oor3I3, simp add: Δ1-defs)
    by (metis prem(1) prem(2) ss)

```

```

next
  case spec-resolveO note  $sr_4 = \text{spec-resolveO}$ 
  show ?thesis using stat  $\Delta_2$  prem  $sr_3$   $sr_4$ 
  unfolding ss lcfgs apply-
  apply(frule  $\Delta_2$ -implies) apply (simp add:  $\Delta_2$ -defs  $\Delta_1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta_1$ -defs)
  by (metis prem(1) prem(2) ss)
next
  case spec-resolve note  $sr_4 = \text{spec-resolve}$ 
  show ?thesis using stat  $\Delta_2$  prem  $sr_3$   $sr_4$ 
  unfolding ss lcfgs apply-
  apply(frule  $\Delta_2$ -implies) apply (simp add:  $\Delta_2$ -defs  $\Delta_1$ -defs)
  apply(rule oor3I3, simp add:  $\Delta_1$ -defs)
  by (metis prem(1) prem(2) ss)
qed
qed. . .

qed
qed

```

```

lemma step3: unwindIntoCond  $\Delta_3$  (oor  $\Delta_3$   $\Delta_1$ )
proof(rule unwindIntoCond-simpleI)
  fix  $n$   $w_1$   $w_2$   $ss_3$   $ss_4$  statA  $ss_1$   $ss_2$  statO
  assume  $r$ : reachO  $ss_3$  reachO  $ss_4$  reachV  $ss_1$  reachV  $ss_2$ 
  and  $\Delta_3$ :  $\Delta_3$   $n$   $w_1$   $w_2$   $ss_3$   $ss_4$  statA  $ss_1$   $ss_2$  statO

  obtain  $pstate_3$   $cfg_3$   $cfgs_3$   $ibT_3$   $ibUT_3$   $ls_3$  where  $ss_3$ :  $ss_3 = (pstate_3, cfg_3, cfgs_3,$ 
ibT_3, ibUT_3, ls_3)
  by (cases  $ss_3$ , auto)
  obtain  $pstate_4$   $cfg_4$   $cfgs_4$   $ibT_4$   $ibUT_4$   $ls_4$  where  $ss_4$ :  $ss_4 = (pstate_4, cfg_4, cfgs_4,$ 
ibT_4, ibUT_4, ls_4)
  by (cases  $ss_4$ , auto)
  obtain  $cfg_1$   $ibT_1$   $ibUT_1$   $ls_1$  where  $ss_1$ :  $ss_1 = (cfg_1, ibT_1, ibUT_1, ls_1)$ 
  by (cases  $ss_1$ , auto)
  obtain  $cfg_2$   $ibT_2$   $ibUT_2$   $ls_2$  where  $ss_2$ :  $ss_2 = (cfg_2, ibT_2, ibUT_2, ls_2)$ 
  by (cases  $ss_2$ , auto)
  note  $ss = ss_3$   $ss_4$   $ss_1$   $ss_2$ 

  obtain  $pc_3$   $vs_3$   $avst_3$   $h_3$   $p_3$  where
  lcfgs3: last  $cfgs_3 = \text{Config } pc_3 (\text{State } (Vstore\ vs_3)\ avst_3\ h_3\ p_3)$ 
  by (cases last  $cfgs_3$ ) (metis state.collapse vstore.collapse)
  obtain  $pc_4$   $vs_4$   $avst_4$   $h_4$   $p_4$  where
  lcfgs4: last  $cfgs_4 = \text{Config } pc_4 (\text{State } (Vstore\ vs_4)\ avst_4\ h_4\ p_4)$ 
  by (cases last  $cfgs_4$ ) (metis state.collapse vstore.collapse)
  note  $lcfgs = lcfgs_3$   $lcfgs_4$ 

```

obtain $hh3$ **where** $h3: h3 = \text{Heap } hh3$ **by**(*cases* $h3$, *auto*)
obtain $hh4$ **where** $h4: h4 = \text{Heap } hh4$ **by**(*cases* $h4$, *auto*)
note $hh = h3 \ h4$

have $f1: \neg \text{finalN } ss1$
using $\Delta3$ **unfolding** $ss \ \Delta3\text{-def}$
apply *clarsimp*
by(*frule common-implies, simp*)

have $f2: \neg \text{finalN } ss2$
using $\Delta3$ **unfolding** $ss \ \Delta3\text{-def}$
apply *clarsimp*
by(*frule common-implies, simp*)

have $f3: \neg \text{finalS } ss3$
using $\Delta3$ **unfolding** ss
apply–**apply**(*frule* $\Delta3\text{-implies}$)
using *finalS-if-spec* **by** *force*

have $f4: \neg \text{finalS } ss4$
using $\Delta3$ **unfolding** ss
apply–**apply**(*frule* $\Delta3\text{-implies}$)
using *finalS-if-spec* **by** *force*

note $\text{finals} = f1 \ f2 \ f3 \ f4$
show $\text{finalS } ss3 = \text{finalS } ss4 \wedge \text{finalN } ss1 = \text{finalS } ss3 \wedge \text{finalN } ss2 = \text{finalS } ss4$
using finals **by** *auto*

then show $\text{isIntO } ss3 = \text{isIntO } ss4$ **by** *simp*

then have $\text{lpc3:pcOf } (\text{last } \text{cfs3}) = 7 \vee$
 $\text{pcOf } (\text{last } \text{cfs3}) = 8$
using $\Delta3$ **unfolding** $ss \ \Delta3\text{-defs}$ **by** *simp*

have $\text{sec3}[\text{simp}]: \neg \text{isSecO } ss3$
using $\Delta3$ **unfolding** ss **by** (*simp add:* $\Delta3\text{-defs misSpecL1-def,metis list.size(3)$
n-not-Suc-n)
have $\text{sec4}[\text{simp}]: \neg \text{isSecO } ss4$
using $\Delta3$ **unfolding** ss
by (*simp add:* $\Delta3\text{-defs misSpecL1-def,metis list.size(3)$ *map-is-Nil-conv nat.distinct(1)*)

have $\text{stat}[\text{simp}]: \wedge s3' \ s4' \ \text{statA}'. \ \text{statA}' = \text{sstatA}' \ \text{statA} \ ss3 \ ss4 \implies$
 $\text{validTransO } (ss3, s3') \implies \text{validTransO } (ss4, s4') \implies$
 $(\text{statA} = \text{statA}' \vee \text{statO} = \text{Diff})$

subgoal for $ss3' \ ss4'$
apply (*cases* $ss3$, *cases* $ss4$, *cases* $ss1$, *cases* $ss2$)

```

apply (cases ss3', cases ss4', clarsimp)
using  $\Delta 3$  finals unfolding ss apply clarsimp
apply(simp-all add:  $\Delta 3$ -defs sstatA'-def)
apply(cases statO, simp-all) by (cases statA, simp-all add: newStat-EqI) .

have vs3 xx = vs4 xx using  $\Delta 3$  lcfs unfolding ss  $\Delta 3$ -defs misSpecL1-def
apply clarsimp
by (metis cfs-Suc-zero config.sel(2) list.set-intros(1) state.sel(1) vstore.sel)

then have a1x:(array-loc aa1 (nat (vs4 xx)) avst4) =
      (array-loc aa1 (nat (vs3 xx)) avst3)
using  $\Delta 3$  lcfs unfolding ss  $\Delta 3$ -defs array-loc-def misSpecL1-def
apply clarsimp
by (metis Zero-not-Suc config.sel(2) last-in-set list.size(3) state.sel(2))

have oor2-rule: $\bigwedge$ ss3' ss4'. ss3  $\rightarrow_S$  ss3'  $\implies$  ss4  $\rightarrow_S$  ss4'  $\implies$ 
      (pcOf (last cfs3) = 7  $\rightarrow$  oor  $\Delta 3$   $\Delta 1$   $\infty$  2 2 ss3' ss4' (sstatA'
statA ss3 ss4) ss1 ss2 statO)
       $\wedge$  (pcOf (last cfs3) = 8  $\rightarrow$  oor  $\Delta 3$   $\Delta 1$   $\infty$  1 1 ss3' ss4' (sstatA'
statA ss3 ss4) ss1 ss2 statO)  $\implies$ 
       $\exists w1' < w1. \exists w2' < w2. oor \Delta 3 \Delta 1 \infty w1' w2' ss3' ss4' (sstatA'
statA ss3 ss4) ss1 ss2 statO$ 
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
subgoal for pstate3' cfs3' cfs3' ib3' ls3'
      pstate4' cfs4' cfs4' ib4' ls4'
using lpc3 apply(elim disjE)

subgoal apply(intro exI[of - 2], intro conjI)
subgoal using  $\Delta 3$  unfolding ss  $\Delta 3$ -defs misSpecL1-def apply clarify
by (metis enat-ord-simps(4) numeral-ne-infinity)
apply(intro exI[of - 2], rule conjI)
subgoal using  $\Delta 3$  unfolding ss  $\Delta 3$ -defs misSpecL1-def apply clarify
by (metis enat-ord-simps(4) numeral-ne-infinity)
by simp

subgoal apply(intro exI[of - 1], intro conjI)
subgoal using  $\Delta 3$  unfolding ss  $\Delta 3$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
apply(intro exI[of - 1], rule conjI)
subgoal using  $\Delta 3$  unfolding ss  $\Delta 3$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
by simp . . .

have pstate3:pstate3 = pstate4 using  $\Delta 3$ [unfolded ss  $\Delta 3$ -defs] by fast
have pcc:pcOf (last cfs3) = pcOf (last cfs4) using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded
ss]] by auto

show react (oor  $\Delta 3$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

```

```

show match1 (oor  $\Delta 3$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2 (oor  $\Delta 3$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12 (oor  $\Delta 3$   $\Delta 1$ ) w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-simpleI, simp-all, rule disjI1)
subgoal for ss3' ss4' apply(cases ss3', cases ss4')
  subgoal for pstate3' cfg3' cfs3' ibT3' ibUT3' ls3'
    pstate4' cfg4' cfs4' ibT4' ibUT4' ls4'
  apply-apply(rule oor2-rule, assumption+, intro conjI impI)

subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] by auto
next
  case nonspec-mispred
  then show ?thesis using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] by auto
next
  case spec-mispred
  then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
next
  case spec-Fence
  then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
next
  case spec-resolveI
  then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
next
  case spec-resolveO
  then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
next
  case spec-resolve note sr3 = spec-resolve
  then have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfs4) cfs4  $\neq$  []
using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] unfolding pstate3 by auto
show ?thesis using prem(2)[unfolded ss prem(5)] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using stat  $\Delta 3$  sr3 unfolding ss by (simp add:  $\Delta 3$ -defs)
next
  case nonspec-mispred
  then show ?thesis using stat  $\Delta 3$  sr3 unfolding ss by (simp add:  $\Delta 3$ -defs)
next
  case spec-normal
  then show ?thesis using r4 by auto
next

```

```

    case spec-mispred
    then show ?thesis using r4 by auto
next
    case spec-Fence
    then show ?thesis using r4 by auto
next
    case spec-resolveO
    then show ?thesis using r4 prem pcc by auto
next
    case spec-resolveI
    then show ?thesis using r4 prem pcc by auto
next
    case spec-resolve note sr4 = spec-resolve
    show ?thesis using stat  $\Delta 3$  prem sr3 sr4
    unfolding ss lcfgs apply-
    apply(frul  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs)
    apply(rule oorI2, simp add:  $\Delta 1$ -defs)
    by (smt(verit) prem(1) prem(2) ss)
qed
next
    case spec-normal note sn3 = spec-normal
    show ?thesis
    using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis using stat  $\Delta 3$  lcfgs sn3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using stat  $\Delta 3$  lcfgs sn3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
    next
      case spec-mispred
      then show ?thesis using stat  $\Delta 3$  lcfgs sn3 unfolding ss by (simp add:
 $\Delta 3$ -defs, metis config.sel(1) last-map)
    next
      case spec-Fence
      then show ?thesis using stat  $\Delta 3$  lcfgs sn3 unfolding ss
      by (simp add:  $\Delta 3$ -defs, metis config.sel(1) last-map)
    next
      case spec-resolveO
      then show ?thesis using prem pcc by auto
    next
      case spec-resolveI
      then show ?thesis using prem pcc by auto
    next
      case spec-resolve
      then show ?thesis using stat  $\Delta 3$  lcfgs sn3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
    next

```

```

case spec-normal note sn4 = spec-normal
show ?thesis
apply(intro oorI1)
unfolding ss  $\Delta 3$ -def prem(4,5) apply– apply(clarify,intro conjI)
  subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss hh
    apply– apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
    using cases-14[of pc3] apply simp apply(elim disjE)
    apply simp-all by (metis config.sel(2) last-in-set state.sel(2) Dist-ignore
a1x )+
  subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss prem(4,5)
hh
    apply– apply(frule  $\Delta 3$ -implies) apply(simp-all add:  $\Delta 3$ -defs)
    using cases-14[of pc3] apply simp apply(elim disjE)
    apply simp-all
    by (metis config.collapse config.inject last-in-set state.sel(1) vstore.sel) +
  subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss prem(4,5)
hh
    apply– apply(frule  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs)
    subgoal using stat  $\Delta 3$  lcfgs prem(1,2) sn3 sn4 unfolding ss hh
    apply– apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
    using cases-14[of pc3] apply simp apply(elim disjE)
    by simp-all
    subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
    apply– apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-14[of pc3] apply (simp add: array-loc-def) apply(elim disjE)
  by (simp-all add: array-loc-def)
  subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
  apply– apply(frule  $\Delta 3$ -implies) apply(simp add:  $\Delta 3$ -defs)
using cases-14[of pc3] apply (simp add: array-loc-def) apply(elim disjE)
  by (simp-all add: array-loc-def)
  subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 unfolding ss hh
  apply– apply(frule  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs)
  subgoal using stat  $\Delta 3$  lcfgs sn3 sn4 prem(6) unfolding ss hh
  apply– apply(frule  $\Delta 3$ -implies) by(simp add:  $\Delta 3$ -defs) .
qed
qed
subgoal premises prem using prem(1)[unfolded ss prem(4)]
proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] by auto
  next
    case nonspec-mispred
      then show ?thesis using  $\Delta 3$ -implies[OF  $\Delta 3$ [unfolded ss]] by auto
    next
      case spec-mispred
        then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
    next
      case spec-Fence

```

```

      then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
    next
      case spec-normal
      then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
    next
      case spec-resolveI
      then show ?thesis using stat  $\Delta 3$  prem(6) unfolding ss by (auto simp
add:  $\Delta 3$ -defs)
    next
      case spec-resolveO note sr3 = spec-resolveO
    show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
    next
      case spec-mispred
      then show ?thesis using prem(6) pcc by auto
    next
      case spec-Fence
      then show ?thesis using prem(6) pcc by auto
    next
      case spec-normal
      then show ?thesis using prem(6) pcc by auto
    next
      case spec-resolveI
      then show ?thesis using prem(6) pcc by auto
    next
      case spec-resolve
      show ?thesis using stat  $\Delta 3$  prem sr3
      unfolding ss lcfgs apply-
      apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      apply (rule oorI2, simp add:  $\Delta 1$ -defs local.spec-resolve)
      by (metis prem(1) ss3)
    next
      case spec-resolveO
      show ?thesis using stat  $\Delta 3$  prem sr3
      unfolding ss lcfgs apply-
      apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
      apply (rule oorI2, simp add:  $\Delta 1$ -defs local.spec-resolveO)
      by (metis prem(1) ss3)
  qed
next
  case spec-resolve note sr3 = spec-resolve

```

```

    show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)
      case nonspec-normal
        then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
      next
        case nonspec-mispred
          then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
        next
          case spec-mispred
            then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
          next
            case spec-Fence
              then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
            next
              case spec-normal
                then show ?thesis using stat  $\Delta 3$  lcfgs sr3 unfolding ss by (simp add:
 $\Delta 3$ -defs)
              next
                case spec-resolveI
                  then show ?thesis using prem(6) pcc by auto
                next
                  case spec-resolve
                    show ?thesis using stat  $\Delta 3$  prem sr3
                    unfolding ss lcfgs apply-
                    apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
                    apply (rule oorI2, simp add:  $\Delta 1$ -defs local.spec-resolve)
                    by (metis prem(1) ss3)
                  next
                    case spec-resolveO
                      show ?thesis using stat  $\Delta 3$  prem sr3
                      unfolding ss lcfgs apply-
                      apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
                      apply (rule oorI2, simp add:  $\Delta 1$ -defs local.spec-resolveO)
                      by (metis prem(1) ss3)
                qed qed . . .
              qed
            qed
          qed

```

```

lemma step4: unwindIntoCond  $\Delta 1'$   $\Delta 1$ 
proof (rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 1'$ :  $\Delta 1'$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

```

```

obtain  $pstate3$   $cfg3$   $cfgs3$   $ibT3$   $ibUT3$   $ls3$  where  $ss3$ :  $ss3 = (pstate3, cfg3, cfgs3,$ 
 $ibT3, ibUT3, ls3)$ 
by (cases  $ss3$ , auto)
obtain  $pstate4$   $cfg4$   $cfgs4$   $ibT4$   $ibUT4$   $ls4$  where  $ss4$ :  $ss4 = (pstate4, cfg4, cfgs4,$ 
 $ibT4, ibUT4, ls4)$ 
by (cases  $ss4$ , auto)
obtain  $cfg1$   $ibT1$   $ibUT1$   $ls1$  where  $ss1$ :  $ss1 = (cfg1, ibT1, ibUT1, ls1)$ 
by (cases  $ss1$ , auto)
obtain  $cfg2$   $ibT2$   $ibUT2$   $ls2$  where  $ss2$ :  $ss2 = (cfg2, ibT2, ibUT2, ls2)$ 
by (cases  $ss2$ , auto)
note  $ss = ss3$   $ss4$   $ss1$   $ss2$ 

obtain  $pc3$   $vs3$   $avst3$   $h3$   $p3$  where
 $cfg3$ :  $cfg3 = Config$   $pc3$  (State (Vstore  $vs3$ )  $avst3$   $h3$   $p3$ )
by (cases  $cfg3$ ) (metis state.collapse vstore.collapse)
obtain  $pc4$   $vs4$   $avst4$   $h4$   $p4$  where
 $cfg4$ :  $cfg4 = Config$   $pc4$  (State (Vstore  $vs4$ )  $avst4$   $h4$   $p4$ )
by (cases  $cfg4$ ) (metis state.collapse vstore.collapse)
note  $cfg = cfg3$   $cfg4$ 

obtain  $hh3$  where  $h3$ :  $h3 = Heap$   $hh3$  by(cases  $h3$ , auto)
obtain  $hh4$  where  $h4$ :  $h4 = Heap$   $hh4$  by(cases  $h4$ , auto)
note  $hh = h3$   $h4$ 

have  $f1$ : $\neg finalN$   $ss1$ 
using  $\Delta 1'$  unfolding  $ss$   $\Delta 1'$ -def
apply clarsimp
by(frule common-implies, simp)

have  $f2$ : $\neg finalN$   $ss2$ 
using  $\Delta 1'$  unfolding  $ss$   $\Delta 1'$ -def
apply clarsimp
by(frule common-implies, simp)

have  $f3$ : $\neg finalS$   $ss3$ 
using  $\Delta 1'$  unfolding  $ss$ 
apply–apply(frule  $\Delta 1'$ -implies)
by (simp add: finalS-while-spec)

have  $f4$ : $\neg finalS$   $ss4$ 
using  $\Delta 1'$  unfolding  $ss$ 
apply–apply(frule  $\Delta 1'$ -implies)
by (simp add: finalS-while-spec)

note  $finals = f1$   $f2$   $f3$   $f4$ 
show  $finalS$   $ss3 = finalS$   $ss4 \wedge finalN$   $ss1 = finalS$   $ss3 \wedge finalN$   $ss2 = finalS$   $ss4$ 
using  $finals$  by auto

```

then show $isIntO\ ss3 = isIntO\ ss4$ **by** *simp*

have *match12-aux*:

```
( $\wedge s1' s2' statA'$ 
   $statA' = sstatA' statA\ ss3\ ss4 \implies$ 
   $validTransO\ (ss3, s1') \implies$ 
   $validTransO\ (ss4, s2') \implies$ 
   $Opt.eqAct\ ss3\ ss4 \implies$ 
  ( $\neg isSecO\ ss3 \wedge \neg isSecO\ ss4 \wedge$ 
    $(statA = statA' \vee statO = Diff) \wedge$ 
    $\Delta1 \infty 1\ 1\ s1'\ s2'\ statA'\ ss1\ ss2\ statO$ )
 $\implies match12\ \Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
apply(rule match12-simpleI, rule disjI1)
```

```
apply(rule exI[of - 1], rule conjI)
  subgoal using  $\Delta1'$  unfolding ss  $\Delta1'$ -defs apply clarify
  by(metis enat-ord-simps(4) infinity-ne-i1)
apply(rule exI[of - 1], rule conjI)
  subgoal using  $\Delta1'$  unfolding ss  $\Delta1'$ -defs apply clarify
  by(metis enat-ord-simps(4) infinity-ne-i1)
by auto
```

show *react* $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$

unfolding *react-def* **proof**(*intro conjI*)

```
show match1  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding match1-def by (simp add: finalS-def final-def)
show match2  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding match2-def by (simp add: finalS-def final-def)
show match12  $\Delta1\ w1\ w2\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
proof(rule match12-aux, intro conjI)
  fix  $ss3'\ ss4'\ statA'$ 
  assume  $statA'$ :  $statA' = sstatA' statA\ ss3\ ss4$ 
  and  $v$ :  $validTransO\ (ss3, ss3')\ validTransO\ (ss4, ss4')$ 
  and  $sa$ :  $Opt.eqAct\ ss3\ ss4$ 
  note  $v3 = v(1)$  note  $v4 = v(2)$ 
```

```
obtain  $pstate3'\ cfg3'\ cfgs3'\ ibT3'\ ibUT3'\ ls3'$  where  $ss3'$ :  $ss3' = (pstate3',$ 
 $cfg3', cfgs3', ibT3', ibUT3', ls3')$ 
by (cases ss3', auto)
obtain  $pstate4'\ cfg4'\ cfgs4'\ ibT4'\ ibUT4'\ ls4'$  where  $ss4'$ :  $ss4' = (pstate4',$ 
 $cfg4', cfgs4', ibT4', ibUT4', ls4')$ 
by (cases ss4', auto)
note  $ss = ss\ ss3'\ ss4'$ 
```

obtain $hh3$ **where** $h3$: $h3 = Heap\ hh3$ **by**(*cases h3, auto*)

obtain $hh4$ **where** $h4$: $h4 = Heap\ hh4$ **by**(*cases h4, auto*)

note $hh = h3\ h4$

```

show  $\neg$  isSecO ss3
  using v sa  $\Delta 1'$  unfolding ss
  by (simp add:  $\Delta 1'$ -defs,metis list.size(3) n-not-Suc-n)

show  $\neg$  isSecO ss4
  using v sa  $\Delta 1'$  unfolding ss
  by (simp add:  $\Delta 1'$ -defs,metis list.size(3) n-not-Suc-n)

show stat: statA = statA'  $\vee$  statO = Diff

  using v sa  $\Delta 1'$ 
  apply (cases ss3, cases ss4, cases ss1, cases ss2)
  apply (cases ss3', cases ss4', clarsimp)
  using v sa  $\Delta 1'$  unfolding ss statA' apply clarsimp
  apply(simp-all add:  $\Delta 1'$ -defs sstatA'-def)
  apply(cases statO, simp-all)
  apply(cases statA, simp-all add: newStat-EqI)
  unfolding finalS-def final-def
  using One-nat-def less-numeral-extra(4)
    less-one list.size(3) map-is-Nil-conv
  by (smt (verit) status.exhaust newStat.simps)

  have pstate3:pstate3 = pstate4 using  $\Delta 1'$ [unfolded ss  $\Delta 1'$ -defs] by fast
  have pc:pcOf (last cfgs3) = pcOf (last cfgs4) using  $\Delta 1'$ [unfolded ss  $\Delta 1'$ -defs]
by auto

  show  $\Delta 1 \infty 1 1$  ss3' ss4' statA' ss1 ss2 statO
    using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-normal
      then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case nonspec-mispred
        then show ?thesis using sa  $\Delta 1'$  stat unfolding ss by (simp add:
 $\Delta 1'$ -defs)
    next
      case spec-Fence
        then show ?thesis using sa  $\Delta 1'$  unfolding ss
          apply (simp add:  $\Delta 1'$ -defs, clarify, elim disjE)
          by (simp-all add:  $\Delta 1'$ -defs  $\Delta 1'$ -defs)
    next
      case spec-mispred
        then show ?thesis using sa  $\Delta 1'$  unfolding ss
          apply (simp add:  $\Delta 1'$ -defs, clarify, elim disjE)
          by (simp-all add:  $\Delta 1'$ -defs  $\Delta 1'$ -defs)
    next
      case spec-normal note sn3 = spec-normal
        show ?thesis using  $\Delta 1'$  sn3(6,7) unfolding ss
          apply (simp add:  $\Delta 1'$ -defs, clarsimp, elim disjE)

```

```

    by (metis is-getInput1 is-Output-1)+
  next
  case spec-resolve note sr3 = spec-resolve
  show ?thesis using v4 [unfolded ss, simplified] proof (cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
    next
    case nonspec-mispred
    then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
    next
    case spec-mispred
    then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
    next
    case spec-normal
    then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
    next
    case spec-Fence
    then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
    next
    case spec-resolve note sr4 = spec-resolve
    show ?thesis
    using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
    apply (simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
    by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
nat-le-linear numeral-le-iff semiring-norm(68,69,72)
length-1-butlast length-map in-set-butlastD)
  next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply (simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
nat-le-linear numeral-le-iff semiring-norm(68,69,72)
length-1-butlast length-map in-set-butlastD)
  next
  case spec-resolveO note sr4 = spec-resolveO
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply (simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
nat-le-linear numeral-le-iff semiring-norm(68,69,72)
length-1-butlast length-map in-set-butlastD)
  qed
  next
  case spec-resolveI note sr3 = spec-resolveI
  show ?thesis using v4 [unfolded ss, simplified] proof (cases rule: stepS-cases)

```

```

    case nonspec-normal
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
next
  case nonspec-mispred
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
next
  case spec-mispred
  then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
next
  case spec-normal
  then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
next
  case spec-Fence
  then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
next
  case spec-resolveO
  then show ?thesis using sr3 pcc  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss],
unfolded whileSpec-defs] by auto
next
  case spec-resolveI note sr4 = spec-resolveI
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply (simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
nat-le-linear numeral-le-iff semiring-norm (68,69,72)
length-1-butlast length-map in-set-butlastD)
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
  apply (simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
  by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
nat-le-linear numeral-le-iff semiring-norm (68,69,72)
length-1-butlast length-map in-set-butlastD)
qed
next
  case spec-resolveO note sr3 = spec-resolveO
  show ?thesis using v4[unfolded ss, simplified] proof (cases rule: stepS-cases)
    case nonspec-normal
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
next
    case nonspec-mispred
  then show ?thesis using  $\Delta 1'$  sr3 unfolding ss by (simp add:  $\Delta 1'$ -defs)
next
    case spec-mispred
  then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded

```

```

whileSpec-defs] by auto
  next
    case spec-normal
      then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
  next
    case spec-Fence
      then show ?thesis using  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss], unfolded
whileSpec-defs] by auto
  next
    case spec-resolveI
      then show ?thesis using sr3 pcc  $\Delta 1'$ -implies[OF  $\Delta 1'$ [unfolded ss],
unfolded whileSpec-defs] by auto
  next
    case spec-resolveO note sr4 = spec-resolveO
      show ?thesis
      using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
      apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
      by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
          nat-le-linear numeral-le-iff semiring-norm(68,69,72)
          length-1-butlast length-map in-set-butlastD)
  next
    case spec-resolve note sr4 = spec-resolve
      show ?thesis
      using sa stat  $\Delta 1'$  v3 v4 sr3 sr4 unfolding ss hh
      apply(simp add:  $\Delta 1'$ -defs  $\Delta 1$ -defs)
      by (metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff empty-set
          nat-le-linear numeral-le-iff semiring-norm(68,69,72)
          length-1-butlast length-map in-set-butlastD)
qed
qed
qed
qed
qed

```

```

lemma step5: unwindIntoCond  $\Delta 2'$   $\Delta 2$ 
proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 2'$ :  $\Delta 2'$  n w1 w2 ss3 ss4 statA ss1 ss2 statO

  obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,
ibT4, ibUT4, ls4)

```

```

by (cases ss4, auto)
obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
by (cases ss1, auto)
obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
by (cases ss2, auto)
note ss = ss3 ss4 ss1 ss2

obtain pc3 vs3 avst3 h3 p3 where
  cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
by (cases cfg3) (metis state.collapse vstore.collapse)
obtain pc4 vs4 avst4 h4 p4 where
  cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
by (cases cfg4) (metis state.collapse vstore.collapse)
note cfg = cfg3 cfg4

obtain hh3 where h3: h3 = Heap hh3 by (cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by (cases h4, auto)
note hh = h3 h4

have f1: ¬finalN ss1
using Δ2' unfolding ss Δ2'-def
apply clarsimp
by (frule common-implies, simp)

have f2: ¬finalN ss2
using Δ2' unfolding ss Δ2'-def
apply clarsimp
by (frule common-implies, simp)

have f3: ¬finalS ss3
  using Δ2' unfolding ss
  apply-apply (frule Δ2'-implies)
  using finalS-while-spec-L2 by force

have f4: ¬finalS ss4
  using Δ2' unfolding ss
  apply-apply (frule Δ2'-implies)
  using finalS-while-spec-L2 by force

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

have sec3[simp]: ¬ isSecO ss3
  using Δ2' unfolding ss

```

```

by (simp add:  $\Delta 2'$ -defs,metis list.size(3) zero-neq-numeral)

have sec4[simp]: $\neg$  isSecO ss4
using  $\Delta 2'$  unfolding ss
by (simp add:  $\Delta 2'$ -defs,metis list.size(3) zero-neq-numeral)

have stat[simp]: $\bigwedge$  s3' s4' statA'. statA' = sstatA' statA ss3 ss4  $\implies$ 
validTransO (ss3, s3')  $\implies$  validTransO (ss4, s4')  $\implies$ 
(statA = statA'  $\vee$  statO = Diff)
subgoal for ss3' ss4'
apply (cases ss3, cases ss4, cases ss1, cases ss2)
apply (cases ss3', cases ss4', clarsimp)
using  $\Delta 2'$  finals unfolding ss apply clarsimp
apply (simp-all add:  $\Delta 2'$ -defs sstatA'-def)
apply (cases statO, simp-all) by (cases statA, simp-all add: newStat-EqI) .

have match12-aux:
( $\bigwedge$  pstate3' cfg3' cfs3' ib3' ibUT3' ls3'
pstate4' cfg4' cfs4' ib4' ibUT4' ls4' statA'.
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)  $\rightarrow$ S (pstate3', cfg3', cfs3', ib3',
ibUT3', ls3')  $\implies$ 
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)  $\rightarrow$ S (pstate4', cfg4', cfs4', ib4',
ibUT4', ls4')  $\implies$ 
Opt.eqAct ss3 ss4  $\implies$  statA' = sstatA' statA ss3 ss4  $\implies$ 
( $\Delta 2 \infty 1 1$  (pstate3', cfg3', cfs3', ib3', ibUT3', ls3') (pstate4', cfg4', cfs4',
ib4', ibUT4', ls4') statA' ss1 ss2 statO))
 $\implies$  match12  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
apply (rule match12-simpleI, simp-all, rule disjI1)

apply (rule exI[of - 1], rule conjI)
subgoal using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
apply (rule exI[of - 1], rule conjI)
subgoal using  $\Delta 2'$  unfolding ss  $\Delta 2'$ -defs apply clarify
by (metis one-less-numeral-iff semiring-norm(76))
subgoal for ss3' ss4' apply (cases ss3', cases ss4')
subgoal for pstate3' cfg3' cfs3' ib3' ibUT3' ls3'
pstate4' cfg4' cfs4' ib4' ibUT4' ls4'
using ss3 ss4 by blast .

have pstate3:pstate3 = pstate4 using  $\Delta 2'$ [unfolded ss  $\Delta 2'$ -defs] by fast

show react  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof (intro conjI)

show match1  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)

```

```

show match2  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12  $\Delta 2$  w1 w2 ss3 ss4 statA ss1 ss2 statO
apply(rule match12-aux)

subgoal premises prem using prem(1)[unfolded ss ]
proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using stat  $\Delta 2'$  unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
    next
    case nonspec-mispred
      then show ?thesis using stat  $\Delta 2'$  unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
      next
      case spec-mispred
        then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
        next
        case spec-normal
          then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
          next
          case spec-Fence
            then show ?thesis using stat  $\Delta 2'$  prem unfolding ss by (auto simp add:
 $\Delta 2'$ -defs)
            next
            case spec-resolve note sr3 = spec-resolve
              have r4:resolve pstate4 (pcOf cfg4 # map pcOf cfgs4) using sr3  $\Delta 2'$ [unfolded
ss  $\Delta 2'$ -defs] pstate3 by auto
              show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
                case nonspec-normal
                  then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
                  next
                  case nonspec-mispred
                    then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
                    next
                    case spec-mispred
                      then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
                      next
                      case spec-normal
                        then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
                        next
                        case spec-Fence
                          then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:

```

```

 $\Delta 2'$ -defs)
  next
    case spec-resolveI note sr4 = spec-resolveI(1,3-) r4
    show ?thesis
    using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
    apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
    apply(intro conjI)
    apply (metis last-map map-butlast map-is-Nil-conv)
    apply (metis image-subset-iff in-set-butlastD)
    apply(metis) apply(metis) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
      list.simps(8) map-L2 map-butlast not-Cons-self2)
    subgoal premises p using p(67-) length2-butlast-eq by fastforce
    subgoal premises p using p(67-) length2-butlast-eq by fastforce .
  next
    case spec-resolveO note sr4 = spec-resolveO
    show ?thesis
    using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
    apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
    apply(intro conjI)
    apply (metis last-map map-butlast map-is-Nil-conv)
    apply (metis image-subset-iff in-set-butlastD)
    apply(metis) apply(metis) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
      list.simps(8) map-L2 map-butlast not-Cons-self2)
    subgoal premises p using p(67-) length2-butlast-eq by fastforce
    subgoal premises p using p(67-) length2-butlast-eq by fastforce .
  next
    case spec-resolve note sr4 = spec-resolve
    show ?thesis
    using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
    apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs)
    apply(intro conjI)
    apply (metis last-map map-butlast map-is-Nil-conv)
    apply (metis image-subset-iff in-set-butlastD)
    apply(metis) apply(metis) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
    apply (metis in-set-butlastD) apply (metis in-set-butlastD)
    apply (smt (verit, cfv-SIG) butlast.simps(2) last-ConsL last-map

```

```

      length-0-conv length-map map-L2 map-butlast not-Cons-self2)
    apply clarify apply (elim disjE)
    apply (metis map-L2 butlast.simps(2) last.simps last-map list.simps(8)

      map-butlast not-Cons-self2 numeral-eq-iff semiring-norm(88))
    using length2-butlast-eq by fastforce+
  qed
next
  case spec-resolveO note sr3 = spec-resolveO
  have r4:is-Output (prog ! pcOf (last cfs4)) using sr3  $\Delta 2'$ -implies[OF
 $\Delta 2'$ [unfolded ss], unfolded whileSpec-defs] by simp
  show ?thesis using prem(2)[unfolded ss prem] proof (cases rule: stepS-cases)
    case nonspec-normal
    then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
    next
    case nonspec-mispred
    then show ?thesis using stat  $\Delta 2'$  sr3 unfolding ss by (simp add:
 $\Delta 2'$ -defs)
    next
    case spec-mispred
    then show ?thesis using r4 by auto
  next
    case spec-normal
    then show ?thesis using r4 by auto
  next
    case spec-Fence
    then show ?thesis using r4 by auto
  next
    case spec-resolveI
    then show ?thesis using r4 by auto
  next
  case spec-resolveO note sr4 = spec-resolveO
  show ?thesis
  using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
  apply (simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)
  apply (intro conjI)
  apply (metis last-map map-butlast map-is-Nil-conv)
  apply (metis image-subset-iff in-set-butlastD)
  apply (metis) apply (metis) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
    list.simps(8) map-L2 map-butlast not-Cons-self2)
  subgoal premises p using p(67-) length2-butlast-eq by fastforce
  subgoal premises p using p(67-) length2-butlast-eq by fastforce .
next

```

```

case spec-resolve note  $sr_4 = \text{spec-resolve}$ 
show ?thesis
using stat  $\Delta 2'$  prem  $sr_3$   $sr_4$  unfolding ss
apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs)
apply(intro conjI)
apply (metis last-map map-butlast map-is-Nil-conv)
apply (metis image-subset-iff in-set-butlastD)
apply(metis) apply(metis) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (smt (verit, ccfv-SIG) butlast.simps(2) last-ConsL last-map
length-0-conv length-map map-L2 map-butlast not-Cons-self2)
apply clarify apply(elim disjE)
apply (metis map-L2 butlast.simps(2) last.simps last-map list.simps(8)

map-butlast not-Cons-self2 numeral-eq-iff semiring-norm(88))
using length2-butlast-eq by fastforce+
qed
next
case spec-resolveI note  $sr_3 = \text{spec-resolveI}$ 
have  $r_4$ :is-getInput (prog ! pcOf (last cfs4)) using  $sr_3$   $\Delta 2'$ -implies[OF
 $\Delta 2'$ [unfolded ss], unfolded whileSpec-defs] by simp
show ?thesis using prem(2)[unfolded ss prem] proof(cases rule: stepS-cases)
case nonspec-normal
then show ?thesis using stat  $\Delta 2'$   $sr_3$  unfolding ss by (simp add:
 $\Delta 2'$ -defs)
next
case nonspec-mispred
then show ?thesis using stat  $\Delta 2'$   $sr_3$  unfolding ss by (simp add:
 $\Delta 2'$ -defs)
next
case spec-mispred
then show ?thesis using  $r_4$  by auto
next
case spec-normal
then show ?thesis using  $r_4$  by auto
next
case spec-Fence
then show ?thesis using  $r_4$  by auto
next
case spec-resolveO
then show ?thesis using  $r_4$  by auto
next
case spec-resolveI note  $sr_4 = \text{spec-resolveI}$ 
show ?thesis
using stat  $\Delta 2'$  prem  $sr_3$   $sr_4$  unfolding ss
apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs, elim conjE)

```

```

apply(intro conjI)
apply (metis last-map map-butlast map-is-Nil-conv)
apply (metis image-subset-iff in-set-butlastD)
apply(metis) apply(metis) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (smt (verit, del-insts) butlast.simps(2) last-ConsL last-map
    list.simps(8) map-L2 map-butlast not-Cons-self2)
subgoal premises p using p(67-) length2-butlast-eq by fastforce
subgoal premises p using p(67-) length2-butlast-eq by fastforce .
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis
  using stat  $\Delta 2'$  prem sr3 sr4 unfolding ss
  apply(simp add:  $\Delta 2'$ -defs  $\Delta 2$ -defs)
  apply(intro conjI)
  apply (metis last-map map-butlast map-is-Nil-conv)
  apply (metis image-subset-iff in-set-butlastD)
  apply(metis) apply(metis) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (metis in-set-butlastD) apply (metis prem(1) prem(2) ss3 ss4)
  apply (metis in-set-butlastD) apply (metis in-set-butlastD)
  apply (smt (verit, ccfv-SIG) butlast.simps(2) last-ConsL last-map
    length-0-conv length-map map-L2 map-butlast not-Cons-self2)
  apply clarify apply(elim disjE)
  apply (metis map-L2 butlast.simps(2) last.simps last-map list.simps(8)
    map-butlast not-Cons-self2 numeral-eq-iff semiring-norm(88))
  using length2-butlast-eq by fastforce+
  qed
qed .
qed
qed

```

```

lemma stepe: unwindIntoCond  $\Delta e \Delta e$ 
proof(rule unwindIntoCond-simpleI)
  fix n w1 w2 ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta e$ :  $\Delta e n w1 w2 ss3 ss4 statA ss1 ss2 statO$ 

  obtain pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfgs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfgs4,

```

```

ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where
  cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
  cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  obtain hh3 where h3: h3 = Heap hh3 by (cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by (cases h4, auto)
  note hh = h3 h4

  show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
    using  $\Delta e$  Opt.final-def finalS-def stepS-endPC endPC-def finalB-endPC
    unfolding  $\Delta e$ -defs ss by clarsimp

  then show isIntO ss3 = isIntO ss4 by simp

  show react  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof (intro conjI)

    show match1  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
    unfolding match1-def by (simp add: finalS-def final-def)
    show match2  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
    unfolding match2-def by (simp add: finalS-def final-def)
    show match12  $\Delta e$  w1 w2 ss3 ss4 statA ss1 ss2 statO
    apply (rule match12-simpleI) using  $\Delta e$  unfolding ss
    by (simp add:  $\Delta e$ -defs stepS-endPC)
  qed
qed

```

lemmas theConds = step0 step1 step2 step3 step4 step5 stepe

proposition lrsecure

proof –

```

define m where m: m  $\equiv$  (7::nat)
define  $\Delta s$  where  $\Delta s$ :  $\Delta s \equiv \lambda i::nat.$ 
  if i = 0 then  $\Delta 0$ 
  else if i = 1 then  $\Delta 1$ 

```

```

else if i = 2 then  $\Delta 2$ 
else if i = 3 then  $\Delta 3$ 
else if i = 4 then  $\Delta 1'$ 
else if i = 5 then  $\Delta 2'$ 
else  $\Delta e$ 
define nxt where nxt: nxt  $\equiv \lambda i::nat.$ 
  if i = 0 then {0,1::nat}
  else if i = 1 then {1,4,2,3,6}
  else if i = 2 then {2,5,1}
  else if i = 3 then {3,1}
  else if i = 4 then {1}
  else if i = 5 then {2}
  else {6}
show ?thesis apply(rule distrib-unwind-lrsecure[of m nxt  $\Delta s$ ])
  subgoal unfolding m by auto
  subgoal unfolding nxt m by auto
  subgoal using init unfolding  $\Delta s$  by auto
  subgoal
    unfolding m nxt  $\Delta s$  apply (simp split: if-splits)
    using theConds
    unfolding oor-def oor3-def oor4-def oor5-def by auto .
qed

end

```

14 Proof of Relative Security for fun2

```

theory Fun-mask
  imports
    ../Instance-IMP/Instance-Secret-IMem
    Relative-Security.Unwinding-fn
begin

```

14.1 Function definition and Boilerplate

```

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

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 ≠ aa2 aa1 ≠ vv aa1 ≠ ii aa1 ≠ temp
  aa2 ≠ aa1 aa2 ≠ vv aa2 ≠ ii aa2 ≠ temp
  vv ≠ aa1 vv ≠ aa2 vv ≠ ii vv ≠ temp
  ii ≠ aa1 ii ≠ aa2 ii ≠ vv ii ≠ temp
  temp ≠ aa1 temp ≠ aa2 temp ≠ vv temp ≠ ii
unfolding vvars-defs by auto

consts size-aa1 :: nat
consts size-aa2 :: nat

fun initVstore :: vstore ⇒ bool where
  initVstore vst = True

fun initAvstore :: avstore ⇒ bool where
  initAvstore (Avstore as) = (as aa1 = (0, size-aa1) ∧ as aa2 = (size-aa1, size-aa2))

definition prog ≡
[
  / Start ,
  / Input T ii ,
  / IfJump (Less (V ii) (N (int (size-aa1)))) 3 7 ,
  / vv ::= (Times (VA aa1 (V ii)) (N 512)) ,
  / 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 ∨ i = 1 ∨ i = 2 ∨ i = 3 ∨ i = 4 ∨ i = 5 ∨
  i = 6 ∨ i = 7 ∨ i = 8 ∨ i > 8
apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
  . . . . .

lemma cases-branch:(i::pcounter) < 3 ∨ i = 3 ∨ i = 4 ∨ i = 5 ∨ i > 5
apply(cases i, simp-all)

```

```

subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
subgoal for i apply(cases i, simp-all)
. . . . .

```

```

lemma ii-aa1-cases: vs ii < int size-aa1 ∨ vs ii ≥ int size-aa1 by auto

```

```

lemma is-If-pcOf[simp]:
pcOf cfg < 8 ⇒ is-IfJump (prog ! (pcOf cfg)) ↔ pcOf cfg = 2
apply(cases cfg) subgoal for pc s using cases-8[of pcOf cfg]
by (auto simp: prog-def) .

```

```

lemma is-If-pc[simp]:
pc < 8 ⇒ is-IfJump (prog ! pc) ↔ pc = 2
using cases-8[of pc]
by (auto simp: prog-def)

```

```

lemma eq-Fence-pc[simp]:
pc < 8 ⇒ prog ! pc ≠ Fence
using cases-8[of pc]
by (auto simp: prog-def)

```

```

lemma not-isInput3[simp]:  $\neg$  is-getInput (prog ! 3)
unfolding prog-def by simp

```

```

lemma not-is-Output[simp]:  $\neg$  is-Output (prog ! 3)
unfolding prog-def by simp

```

```

consts mispred :: predState ⇒ pcounter list ⇒ bool
consts resolve :: predState ⇒ pcounter list ⇒ bool
consts update :: predState ⇒ pcounter list ⇒ predState
consts initPstate :: predState
fun istate :: state ⇒ 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
by (standard, simp add: prog-def)

```

abbreviation

$stepB\text{-abbrev} :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
 $bool$ (**infix** $\rightarrow B$ 55)
where $x \rightarrow B y == stepB\ x\ y$

abbreviation

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

abbreviation

$stepM\text{-abbrev} :: config \times val\ llist \times val\ llist \Rightarrow config \times val\ llist \times val\ llist \Rightarrow$
 $bool$ (**infix** $\rightarrow M$ 55)
where $x \rightarrow M y == stepM\ x\ y$

abbreviation

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

abbreviation

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

abbreviation

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

lemma $endPC[simp]: endPC = 8$

unfolding $endPC\text{-def}$ **unfolding** $prog\text{-def}$ **by** $auto$

lemma $isInput1[simp]: prog\ !\ Suc\ 0 = Input\ T\ ii$

unfolding $prog\text{-def}$ **by** $simp$

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

using $cases\text{-8}[of\ pcOf\ cfg]$ **by** $(auto\ simp: prog\text{-def})$

lemma $is\text{-Output}\text{-pcOf}[simp]: pcOf\ cfg < 8 \Longrightarrow is\text{-Output}\ (prog!(pcOf\ cfg)) \longleftrightarrow$

$pcOf\ cfg = 5 \vee pcOf\ cfg = 7$
using *cases-8*[of *pcOf cfg*] **by** (*auto simp: prog-def*)

lemma *is-Output1*[*simp*]: *is-Output* (*prog ! 5*)
unfolding *is-Output-def prog-def* **by** *auto*
lemma *is-Output2*[*simp*]: *is-Output* (*prog ! 7*)
unfolding *is-Output-def prog-def* **by** *auto*

lemma *isSecV-pcOf*[*simp*]:
isSecV (*cfg, ibT, ibUT*) $\longleftrightarrow pcOf\ cfg = 0$
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf*[*simp*]:
isSecO (*pstate, cfg, cfs, ib, ls*) $\longleftrightarrow (pcOf\ cfg = 0 \wedge cfs = [])$
using *isSecO-def* **by** *simp*

lemma *getActV-pcOf*[*simp*]:
 $pcOf\ cfg < 8 \implies$
getActV (*cfg, ibT, ibUT, ls*) =
 (*if pcOf cfg = 1 then lhd ibT else \perp*)
apply(*subst getActV-simps*) **unfolding** *prog-def*
apply *simp*
using *getActV-simps not-is-getTrustedInput-getActV prog-def* **by** *auto*

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
)
apply(*subst getObsV-simps*)
unfolding *prog-def* **apply** *simp*
using *getObsV-simps not-is-Output-getObsV is-Output-pcOf prog-def* **by** *presburger*

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*)
apply(*subst getActO-simps*)
apply(*cases cfs, auto*)
using *getActV-simps getActV-pcOf prog-def* **by** *presburger*

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  $\perp$ 
)
apply(subst getObsO-simps)
apply(cases cfs, auto)
subgoal unfolding prog-def by auto
subgoal unfolding prog-def by auto
using getObsV-simps is-Output-pcOf not-is-Output-getObsV prog-def by presburger

```

```

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)
unfolding eqSec-def by simp

```

```

lemma nextB-pc0[simp]:
nextB (Config 0 s, ibT, ibUT) =
  (Config 1 s, ibT, ibUT)
apply(subst nextB-Start-Skip-Fence)
unfolding endPC-def unfolding prog-def by auto

```

```

lemma readLocs-pc0[simp]:
readLocs (Config 0 s) = {}
unfolding endPC-def readLocs-def unfolding prog-def by auto

```

```

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)
apply(subst nextB-getTrustedInput')
unfolding endPC-def unfolding prog-def by auto

```

```

lemma readLocs-pc1[simp]:
readLocs (Config 1 s) = {}
unfolding endPC-def readLocs-def unfolding prog-def by auto

```

```

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)

```

apply(subst nextB-getTrustedInput')
unfolding endPC-def **unfolding** prog-def **by** auto

lemma readLocs-pc1 [simp]:
readLocs (Config (Suc 0) s) = {}
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

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)
apply(subst nextB-IfTrue)
unfolding endPC-def **unfolding** prog-def **by** auto

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)
apply(subst nextB-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

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)
by(cases vs ii < int size-aa1, auto)

lemma readLocs-pc2 [simp]:
readLocs (Config 2 s) = {}
unfolding endPC-def readLocs-def **unfolding** prog-def **by** auto

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)
apply(subst nextM-IfTrue)
unfolding endPC-def **unfolding** prog-def **by** auto

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)
apply(subst nextM-IfFalse)
unfolding endPC-def **unfolding** prog-def **by** auto

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)
by(*cases vs ii < int size-aa1, auto*)

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*)
apply(*subst nextB-Assign*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc3[simp]*:
readLocs (*Config 3* (*State* (*Vstore vs*) *avst h p*)) = {*array-loc aa1* (*nat* (*vs ii*))
avst}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-MaskTrue*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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*)
apply(*subst nextB-MaskFalse*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

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}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc4-False[simp]*:
vs ii \geq int size-aa1 \implies
readLocs (*Config 4* (*State* (*Vstore vs*) *avst h p*)) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc5[simp]*:
nextB (Config 5 s, ibT, ibUT) =
(Config 6 s, ibT, ibUT)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc5[simp]*:
readLocs (Config 5 s) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc6[simp]*:
nextB (Config 6 s, ibT, ibUT) = (Config 8 s, ibT, ibUT)
apply(*subst nextB-Jump*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc6[simp]*:
readLocs (Config 6 s) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-pc7[simp]*:
nextB (Config 7 s, ibT, ibUT) = (Config 8 s, ibT, ibUT)
apply(*subst nextB-Output*)
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *readLocs-pc7[simp]*:
readLocs (Config 7 s) = {}
unfolding *endPC-def* *readLocs-def* **unfolding** *prog-def* **by** *auto*

lemma *nextB-stepB-pc*:
 $pc < 8 \implies (pc = 1 \longrightarrow ibT \neq []) \implies$
 $(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow_B nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
apply(*cases s*) **subgoal for** *vst avst hh p* **apply**(*cases vst, cases avst, cases hh*)
subgoal for *vs as h*
using *cases-8[of pc]* **apply** *safe*
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply *simp* **apply**(*subst stepB.simps*) **unfolding** *endPC-def*
apply (*simp add: prog-def*) **by** (*metis llist.collapse*)

subgoal apply(*cases vs ii < int size-aa1*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .
subgoal apply(*cases vs ii < int size-aa1*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .

subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply(*cases vs ii < int size-aa1*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .
subgoal apply(*cases vs ii < int size-aa1*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*) .

subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)
subgoal apply simp apply(*subst stepB.simps*) **unfolding** *endPC-def*
by (*simp add: prog-def*)

subgoal by auto
subgoal by auto

...

lemma not-finalB:
 $pc < 8 \implies (pc = 1 \longrightarrow ibT \neq []) \implies$
 $\neg finalB (Config\ pc\ s, ibT, ibUT)$

using *nextB-stepB-pc* **by** (*simp add: stepB-iff-nextB*)

lemma *finalB-pc-iff'*:

$pc < 8 \implies$

$finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$

$(pc = 1 \wedge ibT = [])$

apply *safe*

subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)

subgoal using *nextB-stepB-pc[of pc]* **by** (*auto simp add: stepB-iff-nextB*)

subgoal using *finalB-iff* **by** *auto* .

lemma *finalB-pc-iff*:

$pc \leq 8 \implies$

$finalB (Config\ pc\ s, ibT, ibUT) \longleftrightarrow$

$(pc = 1 \wedge ibT = [] \vee pc = 8)$

using *cases-8[of pc]* **apply** (*elim disjE, simp add: finalB-def*)

subgoal by (*meson final-def stebB-0*)

by (*simp add: finalB-pc-iff' finalB-endPC*)**+**

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

$pcOf\ cfg \leq 8 \implies$

$finalB (cfg, ibT, ibUT) \longleftrightarrow (pcOf\ cfg = 1 \wedge ibT = [] \vee pcOf\ cfg = 8)$

by (*metis config.collapse finalB-pc-iff*)

lemma *finalS-cond:pcOf cfg < 8 \implies cfigs = [] \implies (pcOf cfg = 1 \longrightarrow ibT \neq LNil)*

$\implies \neg finalS (pstate, cfg, cfigs, ibT, ibUT, ls)$

apply (*cases cfg*)

subgoal for *pc s* **apply** (*cases s*)

subgoal for *vst avst hh p* **apply** (*cases vst, cases hh*)

subgoal for *vs h*

using *cases-8[of pc]* **apply** (*elim disjE*) **unfolding** *finalS-defs*

subgoal using *nonspec-normal[of [] Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT

Config 1 (State (Vstore vs) avst hh p)

ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]

using *is-If-pc* **by** *force*

subgoal apply (*frule nonspec-normal[of cfigs Config pc (State (Vstore vs) avst hh p)*

pstate pstate ibT ibUT

Config 2 (State (Vstore (vs(ii:= lhd ibT))) avst hh p)

lhl ibT ibUT [] ls \cup readLocs (Config pc (State (Vstore

vs) avst hh p)) ls]

prefer 7 **subgoal by** *metis* **by** *simp-all*

subgoal apply (*cases mispred pstate [2]*)

subgoal apply(*frule nonspec-mispred*[of *cfgs Config pc (State (Vstore vs) avst hh p)*]
pstate update pstate [pcOf (Config pc (State (Vstore vs) avst hh p))]
ibT ibUT Config (if vs ii < int size-aa1 then 3 else 7) (State (Vstore vs) avst hh p)
ibT ibUT Config (if vs ii < int size-aa1 then 7 else 3) (State (Vstore vs) avst hh p)
ibT ibUT [Config (if vs ii < int size-aa1 then 7 else 3) (State (Vstore vs) avst hh p)]
ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p)) ls])
prefer 9 subgoal by metis by (*simp add: finalM-iff*)+

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*]
pstate pstate ibT ibUT
Config (if vs ii < int size-aa1 then 3 else 7) (State (Vstore vs) avst hh p)
ibT ibUT [] ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p)) ls])

prefer 7 subgoal by metis by simp-all .

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst hh p)*]
pstate pstate 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 [] ls ∪ readLocs (Config pc (State (Vstore vs) avst hh p)) ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*cases vs ii < int size-aa1*)

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst (Heap h) p)*]
pstate pstate 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 [] insert (array-loc aa2 (nat (vs vv)) avst
ls ls])

prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst (Heap h) p)*]
pstate pstate ibT ibUT
(Config 5 (State (Vstore (vs(temp := 0))) avst (Heap h) p))
ibT ibUT [] ls ls])

prefer 7 subgoal by metis by simp-all .

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc s*
pstate pstate ibT ibUT
Config 6 s
ibT ibUT [] ls ls])
prefer 7 by (*blast,simp-all*)

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst*
hh p)
pstate pstate ibT ibUT
Config 8 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])
prefer 7 subgoal by metis by simp-all

subgoal apply(*frule nonspec-normal*[of *cfgs Config pc (State (Vstore vs) avst*
hh p)
pstate pstate ibT ibUT
Config 8 (State (Vstore vs) avst hh p)
ibT ibUT [] ls ls])
prefer 7 subgoal by metis by simp-all
by simp-all . . .

lemma *not-is-getTrustedInput*[*simp*]:*cfg = Config 3 (State (Vstore vs) (Avstore*
as) (Heap h) p) $\implies \neg is-getInput (prog ! pcOf cfg)$
unfolding *is-getInput-def prog-def by simp*

lemma *notOutput4*[*simp*]: $\neg is-Output (prog ! 4)$ **unfolding prog-def by auto**

lemma *notInput4*[*simp*]: $\neg is-getInput (prog ! 4)$ **unfolding prog-def by auto**

lemma *notInput5*[*simp*]: $\neg is-getInput (prog ! 5)$ **unfolding prog-def by auto**

lemma *finalS-cond-spec*:
pcOf cfg < 8 \implies
((pcOf (last cfgs) = 3 \vee pcOf (last cfgs) = 4 \vee pcOf (last cfgs) = 5) \wedge pcOf
cfg = 7)
 \vee (pcOf (last cfgs) = 7 \wedge pcOf cfg = 3) \implies
length cfgs = Suc 0 \implies
 $\neg finalS (pstate, cfg, cfgs, ibT, ibUT, ls)$
apply(*cases cfg*)
subgoal for pc s apply(*cases s*)
subgoal for vst avst hh p apply(*cases vst, cases avst, cases hh*)
subgoal for vs as h apply(*cases last cfgs*)
subgoal for pcs ss apply(*cases ss*)
subgoal for vsts avsts hhs ps apply(*cases vsts, cases avsts, cases hhs, simp*)
apply(*cases resolve pstate (pcOf cfg # map pcOf cfgs)*)
subgoal unfolding finalS-defs using spec-resolve by (*metis list.size(3) n-not-Suc-n*)
subgoal for vss ass hs apply(*elim disjE, elim conjE, elim disjE, simp*)
unfolding finalS-defs

subgoal apply(*rule notI*,
erule allE[of - (*pstate*, *Config 7* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*),
[*Config 4* (*State* (*Vstore* (*vss*(*vv := hs* (*array-loc aa1* (*nat* (*vss*
ii)) *avsts*)*512))) *avsts hhs ps*]],
ibT, *ibUT*, *ls* \cup *readLocs* (*last cfgs*)))]
by(*erule notE*,
rule spec-normal[of - - - - *Config 4* (*State* (*Vstore* (*vss*(*vv := hs* (*array-loc*
aa1 (*nat* (*vss* *ii*)) *avsts*)*512))) *avsts hhs ps*]], *auto*)

subgoal apply(*cases vss ii < int size-aa1*)
subgoal apply(*rule notI*,
erule allE[of - (*pstate*, *Config 7* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*),
[*Config 5* (*State* (*Vstore* (*vss*(*temp := hs* (*array-loc aa2* (*nat* (*vss*
(*vss vv*)) *avsts*))) *avsts hhs ps*]],
ibT, *ibUT*, *ls* \cup *readLocs* (*last cfgs*)))]
apply(*erule notE*[of (*pstate*, *Config pc* (*State* (*Vstore vs*) (*Avstore as*)
(*Heap h*) *p*), *cfgs*,
ibT, *ibUT*, *ls*) \rightarrow *S*
(*pstate*, *Config 7* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*),
[*Config 5*
(*State* (*Vstore* (*vss*(*temp := hs* (*array-loc aa2* (*nat* (*vss vv*)) *avsts*)))
avsts hhs ps]],
ibT, *ibUT*, *ls* \cup *readLocs* (*last cfgs*)))]
by(*rule spec-normal*, *auto*)
subgoal apply(*rule notI*,
erule allE[of - (*pstate*, *Config 7* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*),
[*Config 5* (*State* (*Vstore* (*vss*(*temp := 0*)) *avsts hhs ps*)],
ibT, *ibUT*, *ls* \cup *readLocs* (*last cfgs*)))]
apply(*erule notE*[of (*pstate*, *Config pc* (*State* (*Vstore vs*) (*Avstore as*)
(*Heap h*) *p*), *cfgs*,
ibT, *ibUT*, *ls*) \rightarrow *S*
(*pstate*, *Config 7* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*),
[*Config 5*
(*State* (*Vstore* (*vss*(*temp := 0*))
avsts hhs ps)],
ibT, *ibUT*, *ls* \cup *readLocs* (*last cfgs*)))]
by(*rule spec-normal*, *auto*) .

subgoal apply(*rule notI*,
erule allE[of - (*update pstate* (*7 # map pcOf cfgs*), *Config 7* (*State* (*Vstore vs*)
(*Avstore as*) (*Heap h*) *p*),
[], *ibT*, *ibUT*, *ls*)]
by(*erule notE*, *rule spec-resolve*, *auto*)

subgoal apply(*rule notI*,
erule allE[of - (*update pstate* (*3 # map pcOf cfgs*), *Config 3* (*State* (*Vstore vs*)
(*Avstore as*) (*Heap h*) *p*),
[], *ibT*, *ibUT*, *ls*)]

```

by(erule notE, rule spec-resolve, auto)
.....

```

end

14.2 Proof

```

theory Fun-mask-secure
  imports Fun-mask
begin

```

```

definition PC ≡ {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 :: stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status ⇒
bool

```

where

```

common = (λ(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
  (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
  statA
  (cfg1, ibT1, ibUT1, ls1)
  (cfg2, ibT2, ibUT2, 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

```

```
(getAstore (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
```

unfolding *common-def PC-def* **by** (*auto simp: image-def subset-eq*)

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 ∧
    ibT3 ≠ [] ∧ ibT4 ≠ [] ∧ ibT1 = ibT3 ∧ ibT2 = ibT4 ∧
    pcOf cfg3 ∈ beforeInput ∧
    ls1 = ls3 ∧ ls2 = ls4 ∧
    noMisSpec cfs3
  ))
```

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

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

```
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO ⇒
(pcOf cfg3 = 1 → ibT3 ≠ LNil) ∧
(pcOf cfg4 = 1 → ibT4 ≠ LNil) ∧
pcOf cfg1 < 8 ∧ pcOf cfg2 = pcOf cfg1 ∧
```

```

cfs3 = [] ∧ pcOf cfs3 < 8 ∧
cfs4 = [] ∧ pcOf cfs4 < 8
unfolding Δ0-defs
apply(intro conjI)
apply (simp-all, linarith+)
apply (metis map-is-Nil-conv)+
by linarith

```

definition Δ1 :: enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status
⇒ bool **where**

```

Δ1 = (λnum
  (pstate3, cfs3, cfs3, ibT3, ibUT3, ls3)
  (pstate4, cfs4, cfs4, ibT4, ibUT4, ls4)
  statA
  (cfs1, ibT1, ibUT1, ls1)
  (cfs2, ibT2, ibUT2, ls2)
  statO.
  (common (pstate3, cfs3, cfs3, ibT3, ibUT3, ls3)
    (pstate4, cfs4, cfs4, ibT4, ibUT4, ls4)
    statA
    (cfs1, ibT1, ibUT1, ls1)
    (cfs2, ibT2, ibUT2, ls2)
    statO ∧
    same-var-o ii cfs3 cfs3 cfs4 cfs4 ∧
    pcOf cfs3 ∈ afterInput ∧
    Dist ls1 ls2 ls3 ls4 ∧
    noMisSpec cfs3
  ))

```

lemmas Δ1-defs = Δ1-def common-def PC-def afterInput-def same-var-o-def noMisSpec-def

lemma Δ1-implies: Δ1 num

```

  (pstate3, cfs3, cfs3, ibT3, ibUT3, ls3)
  (pstate4, cfs4, cfs4, ibT4, ibUT4, ls4)
  statA
  (cfs1, ibT1, ibUT1, ls1)
  (cfs2, ibT2, ibUT2, ls2)
  statO ⇒
  pcOf cfs1 < 8 ∧
  cfs3 = [] ∧ pcOf cfs3 ≠ 1 ∧ pcOf cfs3 < 8 ∧
  cfs4 = [] ∧ pcOf cfs4 ≠ 1 ∧ pcOf cfs4 < 8
unfolding Δ1-defs
apply(intro conjI) apply simp-all
by (metis list.map-disc-iff)

```

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, 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-var-o ii cfg3 cfgs3 cfg4 cfgs4 \wedge
 pcOf cfg3 = startOfThenBranch \wedge cfgs3 \neq [] \wedge
 pcOf (last cfgs3) = startOfElseBranch \wedge
 Dist ls1 ls2 ls3 ls4 \wedge
 misSpecL1 cfgs3
))

lemmas $\Delta 2\text{-defs} = \Delta 2\text{-def}$ *common-def* *PC-def* *same-var-o-def* *startOfThenBranch-def*
startOfElseBranch-def
misSpecL1-def

lemma $\Delta 2\text{-implies}$: $\Delta 2$ 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 (last cfgs3) = 7 \wedge pcOf cfg3 = 3 \wedge
 pcOf (last cfgs4) = pcOf (last cfgs3) \wedge
 pcOf cfg3 = pcOf cfg4 \wedge
 length cfgs3 = Suc 0 \wedge
 length cfgs3 = length cfgs4

apply (intro conjI)
unfolding $\Delta 2\text{-defs}$
apply (simp-all add: image-subset-iff)
by (metis length-map last-map list.map-disc-iff)+

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, 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-var-o ii cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
    vstore (getVstore (stateOf cfg3)) ii  $\geq$  int size-aa1  $\wedge$ 
    pcOf cfg3 = startOfElseBranch  $\wedge$ 
    pcOf (last cfgs3)  $\in$  inThenBranchBeforeOutput  $\wedge$ 
    Dist ls1 ls2 ls3 ls4  $\wedge$ 
    misSpecL1 cfgs3
  ))

```

lemma $\Delta 3$ -def': $\Delta 3$ num

```

  (pstate3,cfg3,cfgs3,ibT3,ibUT3,ls3)
  (pstate4,cfg4,cfgs4,ibT4,ibUT4,ls4)
  statA
  (cfg1,ibT1,ibUT1,ls1)
  (cfg2,ibT2,ibUT2,ls2)
  statO =
  (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-var-o ii cfg3 cfgs3 cfg4 cfgs4  $\wedge$ 
    vstore (getVstore (stateOf cfg3)) ii  $\geq$  int size-aa1  $\wedge$ 
    pcOf cfg3 = startOfElseBranch  $\wedge$ 
    pcOf (last cfgs3)  $\in$  inThenBranchBeforeOutput  $\wedge$ 
    Dist ls1 ls2 ls3 ls4  $\wedge$ 
    misSpecL1 cfgs3
  ) unfolding  $\Delta 3$ -def by auto

```

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

lemma $\Delta 3$ -implies: $\Delta 3$ 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 (last cfgs3) = 3  $\vee$  pcOf (last cfgs3) = 4  $\vee$  pcOf (last cfgs3) = 5)  $\wedge$ 

```

```

pcOf cfg3 = 7 ∧
pcOf (last cfs4) = pcOf (last cfs3) ∧
pcOf cfg3 = pcOf cfg4 ∧
array-base aa1 (getAvstore (stateOf (last cfs3))) = array-base aa1 (getAvstore
(stateOf cfg3)) ∧
array-base aa1 (getAvstore (stateOf (last cfs4))) = array-base aa1 (getAvstore
(stateOf cfg4)) ∧
length cfs3 = Suc 0 ∧
length cfs3 = length cfs4
apply(intro conjI)
unfolding Δ3-defs apply simp-all
apply (simp add: image-subset-iff)
apply (metis last-map map-is-Nil-conv)
apply (metis last-in-set list.size(3) n-not-Suc-n)
apply (metis One-nat-def last-in-set length-0-conv length-map zero-neg-one)
by (metis length-map)

```

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

```

Δe = (λnum
(pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)
(pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)
statA
(cfg1, ibT1, ibUT1, ls1)
(cfg2, ibT2, ibUT2, ls2)
statO.
(pcOf cfg3 = endPC ∧ pcOf cfg4 = endPC ∧ cfs3 = [] ∧ cfs4 = [] ∧
pcOf cfg1 = endPC ∧ pcOf cfg2 = endPC))

```

lemmas $\Delta e\text{-defs} = \Delta e\text{-def}$ *common-def* *endPC*

lemma *init: initCond* $\Delta 0$

```

unfolding initCond-def apply(intro allI)
subgoal for s3 s4 apply(cases s3, cases s4)
subgoal for pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 pstate4 cfg4 cfs4 ibT4 ibUT4 ls4

apply clarify
apply(rule exI[of - (cfg3, ibT3, ibUT3, ls3)])
apply(rule exI[of - (cfg4, ibT4, ibUT4, ls4)])
apply(cases getAvstore (stateOf cfg3), cases getAvstore (stateOf cfg4))
unfolding Δ0-defs
unfolding array-base-def by auto . .

```

```

lemma step0: unwindIntoCond  $\Delta 0$  (oor  $\Delta 0 \Delta 1$ )
proof(rule unwindIntoCond-simpleI)
  fix n ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
  and  $\Delta 0: \Delta 0 n ss3 ss4 statA ss1 ss2 statO$ 

  obtain pstate3 cfg3 cfs3 ibT3 ibUT3 ls3 where ss3: ss3 = (pstate3, cfg3, cfs3,
ibT3, ibUT3, ls3)
  by (cases ss3, auto)
  obtain pstate4 cfg4 cfs4 ibT4 ibUT4 ls4 where ss4: ss4 = (pstate4, cfg4, cfs4,
ibT4, ibUT4, ls4)
  by (cases ss4, auto)
  obtain cfg1 ibT1 ibUT1 ls1 where ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)
  by (cases ss1, auto)
  obtain cfg2 ibT2 ibUT2 ls2 where ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)
  by (cases ss2, auto)
  note ss = ss3 ss4 ss1 ss2

  obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
  obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
  note hh = h3 h4

  have f1:  $\neg$ finalN ss1
    using  $\Delta 0$  finalB-pc-iff' unfolding ss finalN-iff-finalB  $\Delta 0$ -defs
    by simp linarith

  have f2:  $\neg$ finalN ss2
    using  $\Delta 0$  finalB-pc-iff' unfolding ss finalN-iff-finalB  $\Delta 0$ -defs
    by simp linarith

  have f3:  $\neg$ finalS ss3
    using  $\Delta 0$  unfolding ss apply-apply(frule  $\Delta 0$ -implies)
    using finalS-cond by simp

  have f4:  $\neg$ finalS ss4
    using  $\Delta 0$  unfolding ss apply-apply(frule  $\Delta 0$ -implies)
    using finalS-cond by simp

```

```

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react (oor  $\Delta 0$   $\Delta 1$ ) ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

  show match1 (oor  $\Delta 0$   $\Delta 1$ ) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-defs)
  show match2 (oor  $\Delta 0$   $\Delta 1$ ) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-defs)
  show match12 (oor  $\Delta 0$   $\Delta 1$ ) ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI2, intro conjI)
  fix ss3' ss4' statA'
  assume statA': statA' = sstatA' statA ss3 ss4
  and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
  and sa: Opt.eqAct ss3 ss4
  note v3 = v(1) note v4 = v(2)

  obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
  obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
  note ss = ss ss3' ss4'

  obtain pc3 vs3 avst3 h3 p3 where
cfg3: cfg3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)
  by (cases cfg3) (metis state.collapse vstore.collapse)
  obtain pc4 vs4 avst4 h4 p4 where
cfg4: cfg4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)
  by (cases cfg4) (metis state.collapse vstore.collapse)
  note cfg = cfg3 cfg4

  show eqSec ss1 ss3
  using v sa  $\Delta 0$  unfolding ss by (simp add:  $\Delta 0$ -defs)

  show eqSec ss2 ss4
  using v sa  $\Delta 0$  unfolding ss
  apply (simp add:  $\Delta 0$ -defs)
  by (metis length-0-conv length-map)

  show Van.eqAct ss1 ss2
  using v sa  $\Delta 0$  unfolding ss

```

```

unfolding Opt.eqAct-def Van.eqAct-def
apply(simp-all add: Δ0-defs)
using finals map-is-Nil-conv ss
by (metis Δ0 Δ0-implies getActO-pcOf
      numeral-1-eq-Suc-0 numerals(1))

show match12-12 (oor Δ0 Δ1) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
   show sstatO' statO ss1 ss2 = Diff
   using v sa Δ0 sstat unfolding ss cfg statA' apply simp
   apply(simp add: Δ0-defs sstatO'-def sstatA'-def finalS-def final-def)
   using cases-8[of pc3] apply(elim disjE)
   apply simp-all apply(cases statO, simp-all) apply(cases statA, simp-all)
   apply(cases statO, simp-all) apply(cases statA, simp-all)

   by (smt (z3) status.distinct status.exhaust newStat.simps)+
  } note stat = this

show oor Δ0 Δ1 ∞ ss3' ss4' statA' (nextN ss1) (nextN ss2) (sstatO' statO
ss1 ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-mispred
  then show ?thesis using sa Δ0 stat unfolding ss
  apply– apply(frule Δ0-implies)
  by (simp add: Δ0-defs)
  next
  case spec-normal
  then show ?thesis using sa Δ0 stat unfolding ss by (simp add: Δ0-defs)
  next
  case spec-mispred
  then show ?thesis using sa Δ0 stat unfolding ss by (simp add: Δ0-defs)
  next
  case spec-Fence
  then show ?thesis using sa Δ0 stat unfolding ss by (simp add: Δ0-defs)
  next
  case spec-resolve
  then show ?thesis using sa Δ0 stat unfolding ss by (simp add: Δ0-defs)
  next
  case spec-resolveI

```

```

then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
next
  case spec-resolveO
then show ?thesis using sa  $\Delta 0$  stat unfolding ss by (simp add:  $\Delta 0$ -defs)
next
  case nonspec-normal note nn3 = nonspec-normal
  show ?thesis
  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case nonspec-mispred
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-normal
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-mispred
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-Fence
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-resolve
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-resolveI
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case spec-resolveO
    then show ?thesis using sa  $\Delta 0$  stat nn3 unfolding ss by (simp add:
 $\Delta 0$ -defs)
  next
    case nonspec-normal note nn4 = nonspec-normal
  show ?thesis using sa  $\Delta 0$  stat finals v3 v4 nn3 nn4 unfolding ss cfg
apply clarsimp
  apply(cases pc3 = 0)
  subgoal apply(rule oorI1)
  by (simp add:  $\Delta 0$ -defs)
  subgoal apply(subgoal-tac pc4 = 1)
  defer subgoal by (simp add:  $\Delta 0$ -defs)
  apply(rule oorI2)
  by (simp add:  $\Delta 0$ -defs  $\Delta 1$ -defs Opt.eqAct-def) .
qed
qed
qed

```

qed
 qed
 qed

lemma *step1: unwindIntoCond* $\Delta 1$ (*oor4* $\Delta 1$ $\Delta 2$ $\Delta 3$ Δe)

proof(*rule unwindIntoCond-simpleI*)

fix n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

assume r : *reachO* $ss3$ *reachO* $ss4$ *reachV* $ss1$ *reachV* $ss2$

and $\Delta 1$: $\Delta 1$ n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by (*cases* $ss3$, *auto*)

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by (*cases* $ss4$, *auto*)

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (*cases* $ss1$, *auto*)

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (*cases* $ss2$, *auto*)

note $ss = ss3$ $ss4$ $ss1$ $ss2$

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$cfg3$: $cfg3 = Config$ $pc3$ (*State* (*Vstore* $vs3$) $avst3$ $h3$ $p3$)

by (*cases* $cfg3$) (*metis state.collapse vstore.collapse*)

obtain $pc4$ $vs4$ $avst4$ $h4$ $p4$ **where**

$cfg4$: $cfg4 = Config$ $pc4$ (*State* (*Vstore* $vs4$) $avst4$ $h4$ $p4$)

by (*cases* $cfg4$) (*metis state.collapse vstore.collapse*)

note $cfg = cfg3$ $cfg4$

obtain $hh3$ **where** $h3$: $h3 = Heap$ $hh3$ **by**(*cases* $h3$, *auto*)

obtain $hh4$ **where** $h4$: $h4 = Heap$ $hh4$ **by**(*cases* $h4$, *auto*)

note $hh = h3$ $h4$

have $f1$: $\neg finalN$ $ss1$

using $\Delta 1$ *finalB-pc-iff'*

unfolding ss cfg *finalN-iff-finalB* $\Delta 1$ -*defs*

by *simp*

have $f2$: $\neg finalN$ $ss2$

using $\Delta 1$ *finalB-pc-iff'*

unfolding ss cfg *finalN-iff-finalB* $\Delta 1$ -*defs*

by *simp*

have $f3$: $\neg finalS$ $ss3$

using $\Delta 1$ **unfolding** ss **apply-apply**(*frule* $\Delta 1$ -*implies*)

```

using finalS-cond by simp

have f4:¬finalS ss4
  using Δ1 unfolding ss apply-apply(frule Δ1-implies)
  using finalS-cond by simp

note finals = f1 f2 f3 f4

show finalS ss3 = finalS ss4 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
  using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react (oor4 Δ1 Δ2 Δ3 Δe) ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

  show match1 (oor4 Δ1 Δ2 Δ3 Δe) ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 (oor4 Δ1 Δ2 Δ3 Δe) ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 (oor4 Δ1 Δ2 Δ3 Δe) ss3 ss4 statA ss1 ss2 statO

  proof(rule match12-simpleI, rule disjI2, intro conjI)
    fix ss3' ss4' statA'
    assume statA': statA' = sstatA' statA ss3 ss4
    and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
    and sa: Opt.eqAct ss3 ss4
    note v3 = v(1) note v4 = v(2)

    obtain pstate3' cfg3' cfs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
  cfg3', cfs3', ibT3', ibUT3', ls3')
    by (cases ss3', auto)
    obtain pstate4' cfg4' cfs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
  cfg4', cfs4', ibT4', ibUT4', ls4')
    by (cases ss4', auto)
    note ss = ss ss3' ss4'

    show eqSec ss1 ss3
    using v sa Δ1 unfolding ss
    by (simp add: Δ1-defs eqSec-def)

    show eqSec ss2 ss4
    using v sa Δ1 unfolding ss by (simp add: Δ1-defs)

    show Van.eqAct ss1 ss2
    using v sa Δ1 unfolding ss Van.eqAct-def
    by (simp-all add: Δ1-defs)

```

```

show match12-12 (oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta e$ ) ss3' ss4' statA' ss1 ss2 statO
unfolding match12-12-def
proof(rule exI[of - nextN ss1], rule exI[of - nextN ss2], unfold Let-def, intro
conjI impI)
  show validTransV (ss1, nextN ss1)
    by (simp add: f1 nextN-stepN)

  show validTransV (ss2, nextN ss2)
    by (simp add: f2 nextN-stepN)

  {assume sstat: statA' = Diff
  show sstatO' statO ss1 ss2 = Diff
    using v sa  $\Delta 1$  sstat status.distinct(1)
    unfolding ss cfg statA'
  apply(simp add:  $\Delta 1$ -defs sstatO'-def sstatA'-def)
  using cases-8[of pc3] apply(elim disjE)
  defer 1 defer 1
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) map-is-Nil-conv outOf.simps
      unfolding Dist-def by force
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1) newStat.simps by auto
    subgoal apply(cases statO, simp-all) apply(cases statA, simp-all)
      using cfg finals ss status.distinct(1)
      map-is-Nil-conv outOf.simps unfolding Dist-def by force
    by simp+      } note stat = this

  show oor4  $\Delta 1$   $\Delta 2$   $\Delta 3$   $\Delta e$   $\infty$  ss3' ss4' statA' (nextN ss1) (nextN ss2)
(sstatO' statO ss1 ss2)

  using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
    case spec-normal
    then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-mispred
    then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-Fence
    then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next

```

```

      case spec-resolve
    then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-resolveI
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case spec-resolveO
  then show ?thesis using sa  $\Delta 1$  stat unfolding ss by (simp add:  $\Delta 1$ -defs)

  next
    case nonspec-normal note nn3 = nonspec-normal
  show ?thesis using v4 [unfolded ss, simplified] proof (cases rule: stepS-cases)

      case nonspec-mispred
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-normal
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-mispred
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-Fence
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-resolve
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-resolveI
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case spec-resolveO
    then show ?thesis using sa  $\Delta 1$  stat nn3 unfolding ss by (simp add:
 $\Delta 1$ -defs)
  next
    case nonspec-normal note nn4 = nonspec-normal
  then show ?thesis using sa  $\Delta 1$  stat v3 v4 nn3 nn4 unfolding ss cfg hh
apply clarsimp
using cases-8 [of pc3] apply (elim disjE)
  subgoal by (simp add:  $\Delta 1$ -defs)
  subgoal by (simp add:  $\Delta 1$ -defs)

```

```

subgoal using ii-aa1-cases[of vs3] apply(elim disjE)
  subgoal apply(rule oor4I1) apply (simp add: Δ1-defs) .
  subgoal apply(rule oor4I1) apply (simp add: Δ1-defs) . .
subgoal apply(rule oor4I1) by (auto simp add: Δ1-defs)
subgoal using ii-aa1-cases[of vs3] apply(elim disjE)
  subgoal apply(rule oor4I1) by (auto simp add: Δ1-defs)
  subgoal apply(rule oor4I1) by (simp add: Δ1-defs) .
subgoal apply(rule oor4I1) by (simp add: Δ1-defs array-loc-def)
subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δe-defs)
subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δe-defs)
subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δe-defs)
subgoal by (simp add: Δ1-defs Δe-defs) .
qed
next
case nonspec-mispred note nm3 = nonspec-mispred
show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)

  case nonspec-normal
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-normal
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-mispred
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-Fence
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-resolve
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-resolveI
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case spec-resolveO
  then show ?thesis using sa Δ1 stat nm3 unfolding ss by (simp add:
Δ1-defs)
  next
  case nonspec-mispred note nm4 = nonspec-mispred
  then show ?thesis using sa Δ1 stat v3 v4 nm3 nm4 unfolding ss cfg
hh apply clarsimp
  using cases-8[of pc3] apply(elim disjE)

```

```

    subgoal by (simp add:  $\Delta 1$ -defs)
    subgoal by (simp add:  $\Delta 1$ -defs)
    subgoal using ii-aa1-cases[of vs3] apply(elim disjE)
      subgoal apply(rule oor4I2) by (simp add:  $\Delta 1$ -defs  $\Delta 2$ -defs)
      subgoal apply(rule oor4I3) by (simp add:  $\Delta 1$ -defs  $\Delta 3$ -defs) .
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal apply(rule oor4I1) by (simp add:  $\Delta 1$ -defs)
    subgoal by (simp add:  $\Delta 1$ -defs  $\Delta e$ -defs)
    subgoal by (simp add:  $\Delta 1$ -defs  $\Delta e$ -defs) .
  qed
qed
qed
qed
qed
qed

```

lemma *step2: unwindIntoCond $\Delta 2$ $\Delta 1$*

proof(*rule unwindIntoCond-simpleI*)

fix *n ss3 ss4 statA ss1 ss2 statO*

assume *r: reachO ss3 reachO ss4 reachV ss1 reachV ss2*

and $\Delta 2$: $\Delta 2$ *n ss3 ss4 statA ss1 ss2 statO*

obtain *pstate3 cfg3 cfs3 ibT3 ibUT3 ls3* **where** *ss3: ss3 = (pstate3, cfg3, cfs3, ibT3, ibUT3, ls3)*

by (*cases ss3, auto*)

obtain *pstate4 cfg4 cfs4 ibT4 ibUT4 ls4* **where** *ss4: ss4 = (pstate4, cfg4, cfs4, ibT4, ibUT4, ls4)*

by (*cases ss4, auto*)

obtain *cfg1 ibT1 ibUT1 ls1* **where** *ss1: ss1 = (cfg1, ibT1, ibUT1, ls1)*

by (*cases ss1, auto*)

obtain *cfg2 ibT2 ibUT2 ls2* **where** *ss2: ss2 = (cfg2, ibT2, ibUT2, ls2)*

by (*cases ss2, auto*)

note *ss = ss3 ss4 ss1 ss2*

obtain *pc3 vs3 avst3 h3 p3* **where**

lcfs3: last cfs3 = Config pc3 (State (Vstore vs3) avst3 h3 p3)

by (*cases last cfs3*) (*metis state.collapse vstore.collapse*)

obtain *pc4 vs4 avst4 h4 p4* **where**

lcfs4: last cfs4 = Config pc4 (State (Vstore vs4) avst4 h4 p4)

by (*cases last cfs4*) (*metis state.collapse vstore.collapse*)

note *lcfs = lcfs3 lcfs4*

have *f1: \neg finalN ss1*

using $\Delta 2$ *finalB-pc-iff'*

```

unfolding ss finalN-iff-finalB  $\Delta 2$ -defs
by auto

have f2: $\neg$ finalN ss2
using  $\Delta 2$  finalB-pc-iff'
unfolding ss finalN-iff-finalB  $\Delta 2$ -defs
by auto

have f3: $\neg$ finalS ss3
using  $\Delta 2$  unfolding ss
apply-apply(frule  $\Delta 2$ -implies)
using finalS-cond-spec by simp

have f4: $\neg$ finalS ss4
using  $\Delta 2$  unfolding ss apply-apply(frule  $\Delta 2$ -implies)
using finalS-cond-spec by simp

note finals = f1 f2 f3 f4
show finalS ss3 = finalS ss4  $\wedge$  finalN ss1 = finalS ss3  $\wedge$  finalN ss2 = finalS ss4
using finals by auto

then show isIntO ss3 = isIntO ss4 by simp

show react  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
unfolding react-def proof(intro conjI)

show match1  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
unfolding match1-def by (simp add: finalS-def final-def)
show match2  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO
unfolding match2-def by (simp add: finalS-def final-def)
show match12  $\Delta 1$  ss3 ss4 statA ss1 ss2 statO

proof(rule match12-simpleI, rule disjI1, intro conjI)
fix ss3' ss4' statA'
assume statA': statA' = sstatA' statA ss3 ss4
and v: validTransO (ss3, ss3') validTransO (ss4, ss4')
and sa: Opt.eqAct ss3 ss4
note v3 = v(1) note v4 = v(2)

obtain pstate3' cfg3' cfgs3' ibT3' ibUT3' ls3' where ss3': ss3' = (pstate3',
cfg3', cfgs3', ibT3', ibUT3', ls3')
by (cases ss3', auto)
obtain pstate4' cfg4' cfgs4' ibT4' ibUT4' ls4' where ss4': ss4' = (pstate4',
cfg4', cfgs4', ibT4', ibUT4', ls4')
by (cases ss4', auto)
note ss = ss ss3' ss4'

```

```

obtain hh3 where h3: h3 = Heap hh3 by(cases h3, auto)
obtain hh4 where h4: h4 = Heap hh4 by(cases h4, auto)
note hh = h3 h4

```

```

show  $\neg$  isSecO ss3
using v sa  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)

```

```

show  $\neg$  isSecO ss4
using v sa  $\Delta 2$  unfolding ss apply clarsimp
apply (simp add:  $\Delta 2$ -defs) by metis

```

```

show stat: statA = statA'  $\vee$  statO = Diff
using v sa  $\Delta 2$ 
apply (cases ss3, cases ss4, cases ss1)
apply (cases ss2, cases ss3', cases ss4', clarsimp)
using v sa  $\Delta 2$  unfolding ss statA' apply clarsimp
apply(simp-all add:  $\Delta 2$ -defs sstatA'-def)
apply(cases statO, simp-all) apply(cases statA, simp-all)
unfolding finalS-defs
by (smt (verit, ccfv-SIG) newStat.simps(1))

```

```

have isO:is-Output (prog ! pcOf (last cfigs3)) is-Output (prog ! pcOf (last cfigs4)) using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by (simp add: is-Output1)+

```

```

show  $\Delta 1 \infty$  ss3' ss4' statA' ss1 ss2 statO

```

```

using v3[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case nonspec-mispred
      then show ?thesis using sa stat  $\Delta 2$  unfolding ss by (simp add:  $\Delta 2$ -defs)
  next
    case spec-normal
      then show ?thesis using sa stat  $\Delta 2$  v3 unfolding ss apply–
        apply(frule  $\Delta 2$ -implies) by(simp add:  $\Delta 2$ -defs)
  next
    case spec-mispred
      then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply–
        apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
  next
    case spec-Fence
      then show ?thesis using sa stat  $\Delta 2$  unfolding ss apply–
        apply(frule  $\Delta 2$ -implies) by (simp add:  $\Delta 2$ -defs)
  next
    case spec-resolveI
      then show ?thesis using isO by auto
  next

```

```

      case spec-resolve note sr3 = spec-resolve
    show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
      case nonspec-normal
        then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
      next
        case nonspec-mispred
          then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
        next
          case spec-normal
            then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
          next
            case spec-mispred
              then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
            next
              case spec-Fence
                then show ?thesis using sa stat  $\Delta 2$  sr3 unfolding ss by (simp add:
 $\Delta 2$ -defs)
              next
                case spec-resolveI
                  then show ?thesis using isO by auto
                next
                  case spec-resolve note sr4 = spec-resolve
                    show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
                    unfolding ss lcfgs hh apply-
                    apply(frul  $\Delta 2$ -implies)
                    by (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs, clarsimp)
                  next
                    case spec-resolveO note sr4 = spec-resolveO
                      show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
                      unfolding ss lcfgs hh apply-
                      apply(frul  $\Delta 2$ -implies)
                      by (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs, clarsimp)
                    qed
                  next
                    case spec-resolveO note sr3 = spec-resolveO
                      then have cfigs4:cfigs4  $\neq$  [] using  $\Delta 2$ -implies[OF  $\Delta 2$ [unfolded ss]] by auto
                      show ?thesis using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
                        case nonspec-normal
                          then show ?thesis by(simp add:cfigs4)
                        next
                          case nonspec-mispred
                            then show ?thesis by(simp add:cfigs4)
                        next
                          case spec-normal
                            then show ?thesis by (simp add: isO)

```

```

next
  case spec-mispred
  then show ?thesis using isO by auto
next
  case spec-Fence
  then show ?thesis using isO by auto
next
  case spec-resolveI
  then show ?thesis using isO by blast
next
  case spec-resolve note sr4 = spec-resolve
  show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply(frule  $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs, clarsimp)
next
  case spec-resolveO note sr4 = spec-resolveO
  show ?thesis using sa stat  $\Delta 2$  v3 v4 sr3 sr4
  unfolding ss lcfgs hh apply-
  apply(frule  $\Delta 2$ -implies)
  by (simp add:  $\Delta 2$ -defs  $\Delta 1$ -defs, clarsimp)
qed
qed
qed
qed
qed

```

lemma *step3: unwindIntoCond $\Delta 3$ (oor $\Delta 3$ $\Delta 1$)*

proof(rule *unwindIntoCond-simpleI*)

fix n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

assume r : $reachO$ $ss3$ $reachO$ $ss4$ $reachV$ $ss1$ $reachV$ $ss2$

and $\Delta 3$: $\Delta 3$ n $ss3$ $ss4$ $statA$ $ss1$ $ss2$ $statO$

obtain $pstate3$ $cfg3$ $cfgs3$ $ibT3$ $ibUT3$ $ls3$ **where** $ss3$: $ss3 = (pstate3, cfg3, cfgs3, ibT3, ibUT3, ls3)$

by (cases $ss3$, auto)

obtain $pstate4$ $cfg4$ $cfgs4$ $ibT4$ $ibUT4$ $ls4$ **where** $ss4$: $ss4 = (pstate4, cfg4, cfgs4, ibT4, ibUT4, ls4)$

by (cases $ss4$, auto)

obtain $cfg1$ $ibT1$ $ibUT1$ $ls1$ **where** $ss1$: $ss1 = (cfg1, ibT1, ibUT1, ls1)$

by (cases $ss1$, auto)

obtain $cfg2$ $ibT2$ $ibUT2$ $ls2$ **where** $ss2$: $ss2 = (cfg2, ibT2, ibUT2, ls2)$

by (cases $ss2$, auto)

note $ss = ss3$ $ss4$ $ss1$ $ss2$

obtain $pc3$ $vs3$ $avst3$ $h3$ $p3$ **where**

$lcfgs3$: $last$ $cfgs3 = Config$ $pc3$ (State (Vstore $vs3$) $avst3$ $h3$ $p3$)

by (cases $last$ $cfgs3$) (metis state.collapse vstore.collapse)

```

obtain  $pc_4\ vs_4\ avst_4\ h_4\ p_4$  where
 $lcfgs_4$ :  $last\ cfgs_4 = Config\ pc_4\ (State\ (Vstore\ vs_4)\ avst_4\ h_4\ p_4)$ 
by ( $cases\ last\ cfgs_4$ ) ( $metis\ state.collapse\ vstore.collapse$ )
note  $lcfgs = lcfgs_3\ lcfgs_4$ 

obtain  $hh_3$  where  $h_3: h_3 = Heap\ hh_3$  by( $cases\ h_3, auto$ )
obtain  $hh_4$  where  $h_4: h_4 = Heap\ hh_4$  by( $cases\ h_4, auto$ )
note  $hh = h_3\ h_4$ 

have  $f1:\neg finalN\ ss1$ 
  using  $\Delta_3\ finalB-pc-iff'$ 
  unfolding  $ss\ finalN-iff-finalB\ \Delta_3-defs$ 
  by  $auto$ 

have  $f2:\neg finalN\ ss2$ 
  using  $\Delta_3\ finalB-pc-iff'$ 
  unfolding  $ss\ finalN-iff-finalB\ \Delta_3-defs$ 
  by  $auto$ 

have  $f3:\neg finalS\ ss3$ 
  using  $\Delta_3$  unfolding  $ss$ 
  apply–apply( $frule\ \Delta_3-implies$ )
  using  $finalS-cond-spec$  by  $simp$ 

have  $f4:\neg finalS\ ss4$ 
  using  $\Delta_3$  unfolding  $ss$ 
  apply–apply( $frule\ \Delta_3-implies$ )
  using  $finalS-cond-spec$  by  $simp$ 

note  $finals = f1\ f2\ f3\ f4$ 
show  $finalS\ ss3 = finalS\ ss4 \wedge finalN\ ss1 = finalS\ ss3 \wedge finalN\ ss2 = finalS\ ss4$ 
  using  $finals$  by  $auto$ 

then show  $isIntO\ ss3 = isIntO\ ss4$  by  $simp$ 

show  $react\ (oor\ \Delta_3\ \Delta_1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
unfolding  $react-def$  proof( $intro\ conjI$ )

  show  $match1\ (oor\ \Delta_3\ \Delta_1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match1-def$  by ( $simp\ add:\ finalS-def\ final-def$ )
  show  $match2\ (oor\ \Delta_3\ \Delta_1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  unfolding  $match2-def$  by ( $simp\ add:\ finalS-def\ final-def$ )
  show  $match12\ (oor\ \Delta_3\ \Delta_1)\ ss3\ ss4\ statA\ ss1\ ss2\ statO$ 
  proof( $rule\ match12-simpleI, rule\ disjI1, intro\ conjI$ )
  fix  $ss3'\ ss4'\ statA'$ 
  assume  $statA'$ :  $statA' = sstatA'\ statA\ ss3\ ss4$ 
  and  $v$ :  $validTransO\ (ss3, ss3')\ validTransO\ (ss4, ss4')$ 

```

```

and sa: Opt.eqAct ss3 ss4
note v3 = v(1) note v4 = v(2)

obtain pstate3' cfg3' cfigs3' ibT3' ibUT3' ls3'
  where ss3': ss3' = (pstate3', cfg3', cfigs3', ibT3', ibUT3', ls3')
  by (cases ss3', auto)
obtain pstate4' cfg4' cfigs4' ibT4' ibUT4' ls4'
  where ss4': ss4' = (pstate4', cfg4', cfigs4', ibT4', ibUT4', ls4')
  by (cases ss4', auto)
note ss = ss3 ss4 ss1 ss2 ss3' ss4'

show ¬ isSecO ss3
using v sa Δ3 unfolding ss by (simp add: Δ3-defs)

show ¬ isSecO ss4
using v sa Δ3 unfolding ss by (simp add: Δ3-defs)

show stat: statA = statA' ∨ statO = Diff
using v sa Δ3
apply (cases ss3, cases ss4, cases ss1)
apply (cases ss2, cases ss3', cases ss4', clarsimp)
using v sa Δ3 unfolding ss statA' apply clarsimp
apply (simp-all add: Δ3-defs sstatA'-def) apply (cases statO, simp-all)
apply (cases statA, simp-all)
unfolding finalS-defs
by (smt (z3) list.size(3) map-eq-imp-length-eq
      n-not-Suc-n status.exhaust newStat.simps)

have notI-if-fence: ¬ is-getInput (prog ! pcOf (last cfigs4)) ¬ is-IfJump (prog !
pcOf (last cfigs4))
  prog ! pcOf (last cfigs3) ≠ Fence
  using is-Output-pcOf is-getTrustedInput-pcOf
    Δ3-implies[OF Δ3[unfolded ss]] prog-def by auto
  have pc:pcOf (last cfigs4) = pcOf (last cfigs3) cfigs3 ≠ [] cfigs4 ≠ [] using
Δ3-implies[OF Δ3[unfolded ss]] by auto

have vs-eq:vstore (getVstore (stateOf cfg3)) ii = vs3 ii
  vs4 ii = vs3 ii
  using Δ3[unfolded ss Δ3-def' same-var-o-def misSpecL1-def]
    last-in-set[OF pc(2), unfolded lcfgs] last-in-set[OF pc(3), unfolded lcfgs]
  by fastforce+

hence condition:int size-aa1 ≤ vs3 ii int size-aa1 ≤ vs4 ii using vs-eq
Δ3[unfolded ss Δ3-def'] by auto

show oor Δ3 Δ1 ∞ ss3' ss4' statA' ss1 ss2 statO
using v3[unfolded ss, simplified] proof (cases rule: stepS-cases)
  case nonspec-normal

```

```

    then show ?thesis using sa stat  $\Delta 3$  lcfgs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
  next
    case nonspec-mispred
    then show ?thesis using sa stat  $\Delta 3$  lcfgs unfolding ss by (simp-all add:
 $\Delta 3$ -defs)
  next
    case spec-mispred
    then show ?thesis using notI-if-fence pc by auto
  next
    case spec-resolveI
    then show ?thesis using notI-if-fence pc by auto
  next
    case spec-Fence
    then show ?thesis using notI-if-fence pc by auto
  next
    case spec-normal note sn3 = spec-normal
    show ?thesis
    using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
      case nonspec-normal
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case nonspec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
      apply (simp add:  $\Delta 3$ -defs)
      by (metis config.sel(1) last-map)
    next
      case spec-Fence
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
      apply (simp add:  $\Delta 3$ -defs)
      by (metis config.sel(1) last-map)
    next
      case spec-resolve
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sn3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-resolveO
      then show ?thesis using sn3 pc by auto
    next
      case spec-resolveI
      then show ?thesis using sn3 pc by auto
    next
      case spec-normal note sn4 = spec-normal
      show ?thesis apply(rule oorI1)

```

```

using cases-branch[of pc3] apply (elim disjE)
subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
  apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)

subgoal unfolding ss  $\Delta 3$ -def' apply- apply (intro conjI)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs)
    using cases-branch[of pc3] apply simp apply (elim disjE)
    apply simp-all
    by (metis config.sel(2) empty-iff last-in-set length-1-butlast length-map
set-empty2 state.sel(2))+
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs)
    using cases-branch[of pc3] apply simp apply (elim disjE)
    apply simp-all
    by (metis config.sel(2) last-in-set state.sel(1) subset-code(1)
vstore.sel)+
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs ar-
ray-loc-def)
    by (metis Dist-ignore config.sel(2) last-in-set state.sel(1) vstore.sel)+
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs).

subgoal unfolding ss  $\Delta 3$ -def' apply- apply (intro conjI)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 condition unfolding ss
hh
    apply- apply (frule  $\Delta 3$ -implies) apply (simp add:  $\Delta 3$ -defs)
    by (metis config.sel(2) last-in-set state.sel(2))
  subgoal using vs-eq sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 condition unfolding
ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 condition unfolding ss
hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 condition unfolding ss
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)
  subgoal using sa stat  $\Delta 3$  lcfs v3 v4 sn3 sn4 condition unfolding ss
hh
    apply- apply (frule  $\Delta 3$ -implies) by (simp add:  $\Delta 3$ -defs)

```

```

    apply- apply(frul Δ3-implies) by(simp add: Δ3-defs) .
    subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 is-Output1 unfolding
ss hh
    by(simp add: Δ3-defs)
    subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
    apply- apply(frul Δ3-implies) by(simp add: Δ3-defs) .
qed
next
case spec-resolve note sr3 = spec-resolve
show ?thesis
using v4[unfolded ss, simplified] proof(cases rule: stepS-cases)
  case nonspec-normal
  then show ?thesis using sa stat Δ3 lcfgs sr3 unfolding ss
  by (simp add: Δ3-defs)
next
  case nonspec-mispred
  then show ?thesis using sa stat Δ3 lcfgs sr3 unfolding ss
  by (simp add: Δ3-defs)
next
  case spec-mispred
  then show ?thesis using sa stat Δ3 lcfgs sr3 unfolding ss
  by (simp add: Δ3-defs)
next
  case spec-normal
  then show ?thesis using sa stat Δ3 lcfgs sr3 unfolding ss
  by (simp add: Δ3-defs)
next
  case spec-Fence
  then show ?thesis using sa stat Δ3 lcfgs sr3 unfolding ss
  by (simp add: Δ3-defs)
next
case spec-resolveI
then show ?thesis using notI-if-fence pc by auto
next
case spec-resolve note sr4 = spec-resolve
show ?thesis
apply(intro oorI2)
using sa stat Δ3 lcfgs v3 v4 sr3 sr4 unfolding ss hh
apply(simp add: Δ3-defs Δ1-defs)
by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
next
case spec-resolveO note sr4 = spec-resolveO
show ?thesis
apply(intro oorI2)
using sa stat Δ3 lcfgs v3 v4 sr3 sr4 unfolding ss hh
apply(simp add: Δ3-defs Δ1-defs)
by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
qed
next

```

```

case spec-resolveO note  $sr3 = \text{spec-resolveO}$ 
hence isO: is-Output (prog ! pcOf (last cfgs4)) unfolding pc by auto
show ?thesis
using  $v4[\text{unfolded } ss, \text{simplified}]$  proof(cases rule: stepS-cases)
  case nonspec-normal
    then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
    by (simp add:  $\Delta 3$ -defs)
  next
    case nonspec-mispred
      then show ?thesis using sa stat  $\Delta 3$  lcfgs sr3 unfolding ss
      by (simp add:  $\Delta 3$ -defs)
    next
      case spec-mispred
        then show ?thesis using notI-if-fence pc by auto
      next
        case spec-normal
          then show ?thesis using isO by auto
        next
          case spec-Fence
            then show ?thesis using notI-if-fence pc by auto
          next
            case spec-resolveI
              then show ?thesis using notI-if-fence pc by auto
            next
              case spec-resolve note  $sr4 = \text{spec-resolve}$ 
                show ?thesis
                apply(intro oorI2)
                using sa stat  $\Delta 3$  lcfgs v3 v4 sr3 sr4 unfolding ss hh
                apply(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
                by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
              next
                case spec-resolveO note  $sr4 = \text{spec-resolveO}$ 
                  show ?thesis
                  apply(intro oorI2)
                  using sa stat  $\Delta 3$  lcfgs v3 v4 sr3 sr4 unfolding ss hh
                  apply(simp add:  $\Delta 3$ -defs  $\Delta 1$ -defs)
                  by (metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq)
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed

```

```

lemma stepe: unwindIntoCond  $\Delta e$   $\Delta e$ 
proof(rule unwindIntoCond-simpleI)
  fix  $n$   $ss3$   $ss4$  statA  $ss1$   $ss2$  statO
  assume  $r$ : reachO  $ss3$  reachO  $ss4$  reachV  $ss1$  reachV  $ss2$ 

```

```

and  $\Delta e$ :  $\Delta e$   $n$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 

obtain  $pstate3$   $cfg3$   $cfgs3$   $ibT3$   $ibUT3$   $ls3$  where  $ss3$ :  $ss3 = (pstate3, cfg3, cfgs3,$ 
 $ibT3, ibUT3, ls3)$ 
by (cases  $ss3$ , auto)
obtain  $pstate4$   $cfg4$   $cfgs4$   $ibT4$   $ibUT4$   $ls4$  where  $ss4$ :  $ss4 = (pstate4, cfg4, cfgs4,$ 
 $ibT4, ibUT4, ls4)$ 
by (cases  $ss4$ , auto)
obtain  $cfg1$   $ibT1$   $ibUT1$   $ls1$  where  $ss1$ :  $ss1 = (cfg1, ibT1, ibUT1, ls1)$ 
by (cases  $ss1$ , auto)
obtain  $cfg2$   $ibT2$   $ibUT2$   $ls2$  where  $ss2$ :  $ss2 = (cfg2, ibT2, ibUT2, ls2)$ 
by (cases  $ss2$ , auto)
note  $ss = ss3$   $ss4$   $ss1$   $ss2$ 

obtain  $pc3$   $vs3$   $avst3$   $h3$   $p3$  where
   $cfg3$ :  $cfg3 = Config$   $pc3$  (State (Vstore  $vs3$ )  $avst3$   $h3$   $p3$ )
  by (cases  $cfg3$ ) (metis state.collapse vstore.collapse)
obtain  $pc4$   $vs4$   $avst4$   $h4$   $p4$  where
   $cfg4$ :  $cfg4 = Config$   $pc4$  (State (Vstore  $vs4$ )  $avst4$   $h4$   $p4$ )
  by (cases  $cfg4$ ) (metis state.collapse vstore.collapse)
note  $cfg = cfg3$   $cfg4$ 

obtain  $hh3$  where  $h3$ :  $h3 = Heap$   $hh3$  by(cases  $h3$ , auto)
obtain  $hh4$  where  $h4$ :  $h4 = Heap$   $hh4$  by(cases  $h4$ , auto)
note  $hh = h3$   $h4$ 

show  $finalS$   $ss3 = finalS$   $ss4 \wedge finalN$   $ss1 = finalS$   $ss3 \wedge finalN$   $ss2 = finalS$   $ss4$ 
  using  $\Delta e$  Opt.final-def Prog.endPC-def finalS-def stepS-endPC
  unfolding  $\Delta e$ -defs  $ss$  by clarsimp

then show  $isIntO$   $ss3 = isIntO$   $ss4$  by simp

show  $react$   $\Delta e$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 
  unfolding react-def proof(intro conjI)

  show  $match1$   $\Delta e$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 
    unfolding match1-def by (simp add: finalS-def final-def)
  show  $match2$   $\Delta e$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 
    unfolding match2-def by (simp add: finalS-def final-def)
  show  $match12$   $\Delta e$   $ss3$   $ss4$   $statA$   $ss1$   $ss2$   $statO$ 
    apply(rule match12-simpleI)
    using  $\Delta e$  stepS-endPC unfolding  $ss$ 
    by (simp add: \Delta e-defs)
qed
qed

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

```

find-theorems *unwindIntoCond* name: *rsecure*

proposition *rsecure*

proof –

define *m* **where** *m*: *m* \equiv (5::nat)

define Δs **where** Δs : $\Delta s \equiv \lambda i::nat.$

if *i* = 0 *then* $\Delta 0$

else if *i* = 1 *then* $\Delta 1$

else if *i* = 2 *then* $\Delta 2$

else if *i* = 3 *then* $\Delta 3$

else Δe

define *next* **where** *next*: *next* $\equiv \lambda i::nat.$

if *i* = 0 *then* {0,1::nat}

else if *i* = 1 *then* {1,2,3,4}

else if *i* = 2 *then* {1}

else if *i* = 3 *then* {3,1}

else {4}

show *?thesis* **apply**(*rule distrib-unwind-rsecure*[of *m next* Δs])

subgoal **unfolding** *m* **by** *auto*

subgoal **unfolding** *next m* **by** *auto*

subgoal **using** *init* **unfolding** Δs **by** *auto*

subgoal

unfolding *m next* Δs **apply** (*simp split: if-splits*)

using *theConds*

unfolding *oor-def oor3-def oor4-def* **by** *auto* .

qed

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.