A formalized programming language with speculative execution

Jamie Wright

March 17, 2025

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

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 Commands

The language defined by this grammar capture standard basic mechanisms for manipulating scalar and array variables, and (un)conditional jumps, using 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*)
  | *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* ⇒ *avstore* ⇒ *loc* **where**
array-base arr avst ≡ *case avst of* (*Avstore as*) ⇒ *fst* (*as arr*)

definition *array-bound* :: *avname* ⇒ *avstore* ⇒ *nat* **where**

array-bound arr avst ≡ *case avst of* (*Avstore as*) ⇒ *snd* (*as arr*)

definition *array-loc* :: *avname* ⇒ *nat* ⇒ *avstore* ⇒ *loc* **where**
array-loc arr i avst ≡ *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* ⇒ *state* ⇒ *val*
and *bval* :: *bexp* ⇒ *state* ⇒ *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* ≡ *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* ≠ *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* ≠ *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* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒ *bool* (**infix** →*B 55*)
where
Seq-Start-Skip-Fence:
pc < *endPC* ⟹ *prog*!*pc* ∈ {*Start*, *Skip*, *Fence*} ⟹
(*Config pc s*, *ibT*, *ibUT*) →*B* (*Config* (*Suc pc*) *s*, *ibT*, *ibUT*)
|
Assign:
pc < *endPC* ⟹ *prog*!*pc* = (*x* ::= *a*) ⟹
s = *State* (*Vstore vs*) *avst h p* ⟹
(*Config pc s*, *ibT*, *ibUT*)
→*B*
(*Config* (*Suc pc*) (*State* (*Vstore* (*vs*(*x* := *aval a s*))) *avst h p*), *ibT*, *ibUT*)
|
ArrAssign:
pc < *endPC* ⟹ *prog*!*pc* = (*arr*[*index*] ::= *a*) ⟹
v = *aval index s* ⟹ *w* = *aval a s* ⟹
0 ≤ *v* ⟹ *v* < *int* (*array-bound arr avst*) ⟹
l = *array-loc arr* (*nat v*) *avst* ⟹
s = *State vst avst* (*Heap h*) *p*

$\Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT)$
$\rightarrow B$
$(Config\ (Suc\ pc)\ (State\ vst\ avst\ (Heap\ (h(l := w)))\ p),\ ibT,\ ibUT)$
|
getTrustedInput:
$pc < endPC \Longrightarrow prog!pc = Input\ T\ x \Longrightarrow$
$(Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ LCons\ i\ ibT,\ ibUT)$
$\rightarrow B$
$(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := i)))\ avst\ h\ p),\ ibT,\ ibUT)$
|
getUntrustedInput:
$pc < endPC \Longrightarrow prog!pc = Input\ U\ x \Longrightarrow$
$(Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p),\ ibT,\ LCons\ i\ ibUT)$
$\rightarrow B$
$(Config\ (Suc\ pc)\ (State\ (Vstore\ (vs(x := i)))\ avst\ h\ p),\ ibT,\ ibUT)$
|
Output:
$pc < endPC \Longrightarrow prog!pc = Output\ t\ aexp \Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT)$
$\rightarrow B$
$(Config\ (Suc\ pc)\ s,\ ibT,\ ibUT)$
|
Jump:
$pc < endPC \Longrightarrow prog!pc = Jump\ pc1 \Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc1\ s,\ ibT,\ ibUT)$
|
IfTrue:
$pc < endPC \Longrightarrow prog!pc = IfJump\ b\ pc1\ pc2 \Longrightarrow$
$bval\ b\ s \Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc1\ s,\ ibT,\ ibUT)$
|
IfFalse:
$pc < endPC \Longrightarrow prog!pc = IfJump\ b\ pc1\ pc2 \Longrightarrow$
$\neg\ bval\ b\ s \Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ (Config\ pc2\ s,\ ibT,\ ibUT)$

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

abbreviation
stepsB :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒ *bool* (**infix** →*B*∗ 55)
where *x* →*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* ∧
(∀ *x i a*. *prog*!*pc* = (*x*[*i*] ::= *a*) ⟶ *aval i s* ≥ *0* ∧
aval i s < *int* (*array-bound x* (*getAvstore s*))) ∧
(∀ *y*. *prog*!*pc* = *Input T y* ⟶ *ibT* ≠ *LNil*) ∧
(∀ *y*. *prog*!*pc* = *Input U y* ⟶ *ibUT* ≠ *LNil*)
⟷
(∃ *cfg′*. (*Config pc s*, *ibT*, *ibUT*) →*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*) **. . .**

lemma *finalB-iff*:
finalB (*Config pc s*, *ibT*, *ibUT*)
⟷
(*pc* ≥ *endPC* ∨
(∃ *x i a*. *prog*!*pc* = (*x*[*i*] ::= *a*) ∧
(¬ *aval i s* ≥ *0* ∨ ¬ *aval i s* < *int* (*array-bound x* (*getAvstore s*)))) ∨
(∃ *y*. *prog*!*pc* = *Input T y* ∧ *ibT* = *LNil*) ∨
(∃ *y*. *prog*!*pc* = *Input U y* ∧ *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 →*B* *cfg-ib′* ⟹ *cfg-ib* →*B* *cfg-ib″* ⟹ *cfg-ib″* = *cfg-ib′*
apply(*induction arbitrary*: *cfg-ib″ rule*: *stepB.induct*)
by (*auto elim*: *stepB.cases*)

definition *nextB* :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist*
where
nextB cfg-ib ≡ *SOME cfg′-ib′*. *cfg-ib* →*B* *cfg′-ib′*

lemma *nextB-stepB*: ¬ *finalB cfg-ib* ⟹ *cfg-ib* →*B* (*nextB cfg-ib*)
unfolding *nextB-def* **apply**(*rule someI-ex*)
unfolding *finalB-def final-def* **by** *auto*

lemma *stepB-nextB*: *cfg-ib* →*B* *cfg′-ib′* ⟹ *cfg′-ib′* = *nextB cfg-ib*
unfolding *nextB-def* **apply**(*rule sym*) **apply**(*rule some-equality*)
using *stepB-determ* **by** *auto*

lemma *nextB-iff-stepB*: ¬ *finalB cfg-ib* ⟹ *nextB cfg-ib* = *cfg′-ib′* ⟷ *cfg-ib* →*B* *cfg′-ib′*
using *nextB-stepB stepB-nextB* **by** *blast*

lemma *stepB-iff-nextB*: *cfg-ib* →*B* *cfg′-ib′* ⟷ ¬ *finalB cfg-ib* ∧ *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* ⟹ *prog*!*pc* ∈ {*Start*, *Skip*, *Fence*} ⟹
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* ⟹ *prog*!*pc* = (*x* ::= *a*) ⟹
s = *State* (*Vstore vs*) *avst h p* ⟹
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* ⟹ *prog*!*pc* = (*arr*[*index*] ::= *a*) ⟹
ls′ = *readLocs a vst avst* (*Heap h*) ⟹
v = *aval index s* ⟹ *w* = *aval a s* ⟹
0 ≤ *v* ⟹ *v* < *int* (*array-bound arr avst*) ⟹
l = *array-loc arr* (*nat v*) *avst* ⟹
s = *State vst avst* (*Heap h*) *p*

$\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = (*Input T x*) $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = (*Input U x*) $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = *Input T x* $\Longrightarrow$
ibT ≠ *LNil* $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = *Input U x* $\Longrightarrow$
ibUT ≠ *LNil* $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = *Output t aexp* $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = *Jump pc1* $\Longrightarrow$
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* ⟹ *prog*!*pc* = *IfJump b pc1 pc2* ⟹
bval b s ⟹
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* ⟹ *prog*!*pc* = *IfJump b pc1 pc2* ⟹
¬ *bval b s* ⟹
nextB (*Config pc s*, *ibT*, *ibUT*) = (*Config pc2 s*, *ibT*, *ibUT*)
by(*intro stepB-nextB*[*THEN sym*] *stepB.intros*)

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

lemma *stepB-endPC*: *pcOf cfg* = *endPC* ⟹ ¬ (*cfg*, *ibT*, *ibUT*) →*B* (*cfg*′, *ibT*′,*ibUT*′)
by (*simp add*: *stepB-iff-nextB finalB-endPC*)

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

lemma *stebB-0*: (*Config 0 s*, *ibT*, *ibUT*) →*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. T he following are some useful elimination rules to help simplify reasoning

lemma *stepB-Seq-Start-Skip-FenceE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg*′, *ibT*′,*ibUT*′)›
and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg*′ = (*Config pc*′ (*State* (*Vstore vs*′) *avst*′ *h*′ *p*′))›
and ‹*prog*!*pc* ∈ {*Start*, *Skip*, *Fence*}›
shows ‹*vs*′ = *vs* ∧ *ibT* = *ibT*′ ∧ *ibUT* = *ibUT*′ ∧
pc′ = *Suc pc* ∧ *avst*′ = *avst* ∧ *h*′ = *h* ∧
p′ = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg*′, *ibT*′,*ibUT*′) *rule*: *stepB.cases*)
by *auto*

lemma *stepB-AssignE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg*′, *ibT*′,*ibUT*′)›

and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*))›
and ‹*prog*!*pc* = (*x* ::= *a*)›
shows ‹*vs'* = (*vs*(*x* := *aval a* (*stateOf cfg*))) ∧
ibT = *ibT'* ∧ *ibUT* = *ibUT'* ∧ *pc'* = *Suc pc* ∧
avst' = *avst* ∧ *h'* = *h* ∧ *p'* = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*,*ibUT'*) *rule*: *stepB.cases*)
by *auto*

lemma *stepB-getTrustedInputE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg'*, *ibT'*,*ibUT'*)›
and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*))›
and ‹*prog*!*pc* = *Input T x*›
shows ‹*vs'* = (*vs*(*x* := *lhd ibT*)) ∧
ibT' = *ltl ibT* ∧ *ibUT* = *ibUT'* ∧ *pc'* = *Suc pc* ∧
avst' = *avst* ∧ *h'* = *h* ∧ *p'* = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*,*ibUT'*) *rule*: *stepB.cases*)
by *auto*

lemma *stepB-getUntrustedInputE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg'*, *ibT'*,*ibUT'*)›
and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*))›
and ‹*prog*!*pc* = *Input U x*›
shows ‹*vs'* = (*vs*(*x* := *lhd ibUT*)) ∧
ibT' = *ibT* ∧ *ibUT'* = *ltl ibUT* ∧ *pc'* = *Suc pc* ∧
avst' = *avst* ∧ *h'* = *h* ∧ *p'* = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*,*ibUT'*) *rule*: *stepB.cases*)
by *auto*

lemma *stepB-OutputE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg'*, *ibT'*,*ibUT'*)›
and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*))›
and ‹*prog*!*pc* = *Output t aexp*›
shows ‹*vs'* = *vs* ∧ *ibT'* = *ibT* ∧ *ibUT'* = *ibUT* ∧
pc' = *Suc pc* ∧ *avst'* = *avst* ∧ *h'* = *h* ∧ *p'* = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*,*ibUT'*) *rule*: *stepB.cases*)
by *auto*

lemma *stepB-JumpE*:
assumes ‹(*cfg*, *ibT*, *ibUT*) →*B* (*cfg'*, *ibT'*,*ibUT'*)›
and ‹*cfg* = (*Config pc* (*State* (*Vstore vs*) *avst h p*))›
and ‹*cfg'* = (*Config pc'* (*State* (*Vstore vs'*) *avst' h' p'*))›
and ‹*prog*!*pc* = *Jump pc1*›
shows ‹*vs'* = *vs* ∧ *ibT'* = *ibT* ∧ *ibUT'* = *ibUT* ∧
pc' = *pc1* ∧ *avst'* = *avst* ∧ *h'* = *h* ∧ *p'* = *p*›
using *assms* **apply** (*cases* (*cfg*, *ibT*, *ibUT*) (*cfg'*, *ibT'*,*ibUT'*) *rule*: *stepB.cases*)

by *auto*

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

lemma *stepB-IfFalseE*:
assumes ‹$(cfg, ibT, ibUT) \to B\ (cfg', ibT', ibUT')$›
and ‹$cfg = (Config\ pc\ (State\ (Vstore\ vs)\ avst\ h\ p))$›
and ‹$cfg' = (Config\ pc'\ (State\ (Vstore\ vs')\ avst'\ h'\ p'))$›
and ‹$prog!pc = IfJump\ b\ pc1\ pc2$› **and** ‹$\neg bval\ b\ (stateOf\ cfg)$›
shows ‹$vs' = vs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge$
$pc' = pc2 \wedge avst' = avst \wedge h' = h \wedge p' = p$›
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 :: $bexp \Rightarrow state \Rightarrow loc\ set$ **where**
$readLocsA\ (N\ n)\ s = \{\}$
|
$readLocsA\ (V\ x)\ s = \{\}$
|
$readLocsA\ (VA\ arr\ index)\ s =$
$insert\ (array\text{-}loc\ arr\ (nat\ (aval\ index\ s))\ (getAvstore\ s))$
$(readLocsA\ index\ s)$
|
$readLocsA\ (Plus\ a1\ a2)\ s = readLocsA\ a1\ s \cup readLocsA\ a2\ s$
|
$readLocsA\ (Times\ a1\ a2)\ s = readLocsA\ a1\ s \cup readLocsA\ a2\ s$
|
$readLocsA\ (Ite\ b\ a1\ a2)\ s = readLocsB\ b\ s \cup readLocsA\ a1\ s \cup readLocsA\ a2\ s$

|
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 a s*
|
readLocsC (*Output t a*) *s* = *readLocsA a s*
|
readLocsC (*IfJump b n1 n2*) *s* = *readLocsB b s*
|
readLocsC - - = {}

context *Prog*
begin

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

end

end

3 Normal Semantics

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

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

theory *Step-Normal*
imports *Step-Basic*
begin

context *Prog*

begin

fun *stepN* :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *loc set* ⇒ *bool* (**infix** ‹→N› *55*)
where
(*cfg*,*ibT*,*ibUT*,*ls*) →*N* (*cfg*′,*ibT*′,*ibUT*′,*ls*′) =
((*cfg*,*ibT*,*ibUT*) →*B* (*cfg*′,*ibT*′,*ibUT*′) ∧ *ls*′ = *ls* ∪ *readLocs cfg*)

abbreviation
stepsN :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *loc set* ⇒ *bool* (**infix** ‹→N∗› *55*)
where *x* →*N*∗ *y* == *star stepN x y*

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

lemma *finalN-iff-finalB*[*simp*]:
finalN (*cfg*, *ibT*, *ibUT*, *ls*) ⟷ *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*′) ⇒ (*cfg*′,*ibT*′,*ibUT*′,*ls* ∪ *readLocs cfg*))

lemma *nextN-stepN*: ¬ *finalN cfg-ib-ls* ⟹ *cfg-ib-ls* →*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-ib-ls* →*N* *cfg*′*-ib*′*-ls*′ ⟹ *cfg*′*-ib*′*-ls*′ = *nextN cfg-ib-ls*
apply(*cases cfg-ib-ls*) **apply**(*cases cfg*′*-ib*′*-ls*′)
using *Prog.stepB-nextB Prog-axioms* **by** *auto*

lemma *nextN-iff-stepN*:
¬ *finalN cfg-ib-ls* ⟹ *nextN cfg-ib-ls* = *cfg*′*-ib*′*-ls*′ ⟷ *cfg-ib-ls* →*N* *cfg*′*-ib*′*-ls*′
using *nextN-stepN stepN-nextN* **by** *blast*

lemma *stepN-iff-nextN*: *cfg-ib-ls* →*N* *cfg*′*-ib*′*-ls*′ ⟷ ¬ *finalN cfg-ib-ls* ∧ *nextN cfg-ib-ls* = *cfg*′*-ib*′*-ls*′
by (*metis finalN-def final-def stepN-nextN*)

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

lemma *stepN-endPC*: *pcOf cfg* = *endPC* ⟹ ¬ (*cfg*,*ibT*,*ibUT*) →*N* (*cfg'*,*ibT'*,*ibUT'*)
  by (*simp add*: *finalN-endPC stepN-iff-nextN*)

lemma *stebN-0*: (*Config 0 s*, *ibT*, *ibUT*, *ls*) →*N* (*Config 1 s*, *ibT*, *ibUT*, *ls*)
using *prog-0 One-nat-def stebB-0* **by** (*auto simp*: *readLocs-def*)

lemma *finalB-eq-finalN*:*finalB* (*cfg*, *ibT*,*ibUT*) ⟷ (∀ *ls*. *finalN* (*cfg*, *ibT*,*ibUT*, *ls*))
  unfolding *finalN-defs finalB-def*
  apply *standard* **by** *auto*

3.2 Elimination Rules

lemma *stepN-Assign2E*:
  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* = (*x* ::= *a*)› **and** ‹*pcOf cfg1* = *pcOf cfg2*›
  shows ‹*vs1'* = (*vs1*(*x* := *aval a* (*stateOf cfg1*))) ∧ *ibT1* = *ibT1'* ∧ *ibUT1* = *ibUT1'* ∧
    vs2' = (*vs2*(*x* := *aval a* (*stateOf cfg2*))) ∧ *ibT2* = *ibT2'* ∧ *ibUT2* = *ibUT2'* ∧
    pc1' = *Suc pc1* ∧ *pc2'* = *Suc pc2* ∧ *ls2'* = *ls2* ∪ *readLocs cfg2* ∧
    avst1' = *avst1* ∧ *avst2'* = *avst2* ∧ *ls1'* = *ls1* ∪ *readLocs cfg1*›
  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* ⇒ *pcounter list* ⇒ *bool*
and *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool*
and *update* :: *predState* ⇒ *pcounter list* ⇒ *predState*
begin

4.2 Mispredicting Step

stepM simply goes the other way than stepB at branches

inductive
stepM :: *config* × *val llist* × *val llist* ⇒ *config* × *val llist* × *val llist* ⇒ *bool* (**infix** →*M* 55)
where
IfTrue[*intro*]:
pc < *endPC* ⟹ *prog*!*pc* = *IfJump b pc1 pc2* ⟹
bval b s ⟹
(*Config pc s*, *ibT*, *ibUT*) →*M* (*Config pc2 s*, *ibT*, *ibUT*)
|
IfFalse[*intro*]:
pc < *endPC* ⟹ *prog*!*pc* = *IfJump b pc1 pc2* ⟹
¬ *bval b s* ⟹
(*Config pc s*, *ibT*, *ibUT*) →*M* (*Config pc1 s*, *ibT*, *ibUT*)

4.2.1 State Transitions

definition *finalM* = *final stepM*

lemma *finalM-iff-aux*:
pc < *endPC* ∧ *is-IfJump* (*prog*!*pc*)
⟷
(∃ *cfg*′. (*Config pc s*, *ibT*, *ibUT*) →*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*,*meson IfFalse IfTrue*) **. .**

lemma *finalM-iff*:
finalM (*Config pc* (*State vst avst h p*), *ibT*, *ibUT*)
⟷
(*pc* ≥ *endPC* ∨ ¬ *is-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*) ⟹ *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*) $\Longrightarrow$ $\neg$ *finalB* (*cfg*, *ibT*, *ibUT*)
using *finalB-imp-finalM* **by** *blast*

lemma *stepM-determ*:
cfg-ib $\rightarrow$*M* *cfg-ib'* $\Longrightarrow$ *cfg-ib* $\rightarrow$*M* *cfg-ib''* $\Longrightarrow$ *cfg-ib''* = *cfg-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-ib $\equiv$ *SOME cfg'-ib'*. *cfg-ib* $\rightarrow$*M* *cfg'-ib'*

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

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

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

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

lemma *nextM-IfTrue*[*simp*]:
pc < *endPC* $\Longrightarrow$ *prog*!*pc* = *IfJump b pc1 pc2* $\Longrightarrow$
$\neg$ *bval b s* $\Longrightarrow$
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* $\Longrightarrow$ *prog*!*pc* = *IfJump b pc1 pc2* $\Longrightarrow$
bval b s $\Longrightarrow$
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* ⇒ *configS* ⇒ *bool* (**infix** →*S* *55*)
where
nonspec-normal:
cfgs = [] ⟹

¬ is-IfJump (prog!(pcOf cfg)) ∨ ¬ mispred pstate [pcOf cfg] ⟹
pstate′ = pstate ⟹
¬ finalB (cfg, ibT, ibUT) ⟹ (cfg′, ibT′, ibUT′) = nextB (cfg, ibT, ibUT) ⟹
cfgs′ = [] ⟹
ls′ = ls ∪ readLocs cfg
⟹
(pstate, cfg, cfgs, ibT, ibUT, ls) →S (pstate′, cfg′, cfgs′, ibT′, ibUT′, ls′)
|
nonspec-mispred:
cfgs = [] ⟹
is-IfJump (prog!(pcOf cfg)) ⟹ mispred pstate [pcOf cfg] ⟹
pstate′ = update pstate [pcOf cfg] ⟹
¬ finalM (cfg, ibT, ibUT) ⟹ (cfg′, ibT′, ibUT′) = nextB (cfg, ibT, ibUT) ⟹
(cfg1′, ibT1′, ibUT1′) = nextM (cfg, ibT, ibUT) ⟹
cfgs′ = [cfg1′] ⟹
ls′ = ls ∪ readLocs cfg
⟹
(pstate, cfg, cfgs, ibT, ibUT, ls) →S (pstate′, cfg′, cfgs′, ibT′, ibUT′, ls′)
|
spec-normal:
cfgs ≠ [] ⟹
¬ resolve pstate (pcOf cfg # map pcOf cfgs) ⟹
¬ is-IfJump (prog!(pcOf (last cfgs))) ∨ ¬ mispred pstate (pcOf cfg # map pcOf cfgs) ⟹
prog!(pcOf (last cfgs)) ≠ Fence ⟹
pstate′ = pstate ⟹
¬ is-getInput (prog!(pcOf (last cfgs))) ⟹
¬ is-Output (prog!(pcOf (last cfgs))) ⟹
¬ finalB (last cfgs, ibT, ibUT) ⟹ (cfg1′,ibT′, ibUT′) = nextB (last cfgs, ibT, ibUT) ⟹
cfg′ = cfg ⟹ cfgs′ = butlast cfgs @ [cfg1′] ⟹
ls′ = ls ∪ readLocs (last cfgs)
⟹
(pstate, cfg, cfgs, ibT, ibUT, ls) →S (pstate′, cfg′, cfgs′, ibT′, ibUT′, ls′)
|
spec-mispred:
cfgs ≠ [] ⟹
¬ resolve pstate (pcOf cfg # map pcOf cfgs) ⟹
is-IfJump (prog!(pcOf (last cfgs))) ⟹ mispred pstate (pcOf cfg # map pcOf cfgs) ⟹
pstate′ = update pstate (pcOf cfg # map pcOf cfgs) ⟹
¬ finalM (last cfgs, ibT, ibUT) ⟹
(lcfg′, ibT′, ibUT′) = nextB (last cfgs, ibT, ibUT) ⟹ (cfg1′, ibT1′, ibUT1′) = nextM (last cfgs, ibT, ibUT) ⟹
cfg′ = cfg ⟹ cfgs′ = butlast cfgs @ [lcfg′] @ [cfg1′] ⟹
ls′ = ls ∪ readLocs (last cfgs)
⟹
(pstate, cfg, cfgs, ibT, ibUT, ls) →S (pstate′, cfg′, cfgs′, ibT′, ibUT′, ls′)
|

spec-Fence:
$cfgs \neq [] \Longrightarrow$
$\neg$ *resolve pstate* $(pcOf\ cfg\ \#\ map\ pcOf\ cfgs) \Longrightarrow$
$prog!(pcOf\ (last\ cfgs)) = Fence \Longrightarrow$
$pstate' = pstate \Longrightarrow cfg' = cfg \Longrightarrow cfgs' = [] \Longrightarrow$
$ibT = ibT' \Longrightarrow ibUT = ibUT' \Longrightarrow ls' = ls$
$\Longrightarrow$
$(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \to S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$
|
spec-resolve:
$cfgs \neq [] \Longrightarrow$
resolve pstate $(pcOf\ cfg\ \#\ map\ pcOf\ cfgs) \Longrightarrow$
$pstate' = update\ pstate\ (pcOf\ cfg\ \#\ map\ pcOf\ cfgs) \Longrightarrow$
$cfg' = cfg \Longrightarrow cfgs' = butlast\ cfgs \Longrightarrow$
$ibT = ibT' \Longrightarrow ibUT = ibUT' \Longrightarrow ls' = ls$
$\Longrightarrow$
$(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \to S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$

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

4.3.1 State Transitions

lemma *stepS-nonspec-normal-iff* [*simp*]:
$cfgs = [] \Longrightarrow \neg\ is\text{-}IfJump\ (prog!(pcOf\ cfg)) \lor \neg\ mispred\ pstate\ [pcOf\ cfg]$
$\Longrightarrow$
$(pstate,\ cfg,\ cfgs,\ ibT,\ ibUT,\ ls) \to S\ (pstate',\ cfg',\ cfgs',\ ibT',\ ibUT',\ ls')$
$\longleftrightarrow$
$(pstate' = pstate \land \neg\ finalB\ (cfg,\ ibT,\ ibUT) \land$
$(cfg',\ ibT',\ ibUT') = nextB\ (cfg,\ ibT,\ ibUT) \land$
$cfgs' = [] \land ls' = ls \cup readLocs\ cfg)$
apply(*subst stepS.simps*) **by** *auto*

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

lemma *stepS-nonspec-mispred-iff* [*simp*]:
$cfgs = [] \Longrightarrow is\text{-}IfJump\ (prog!(pcOf\ cfg)) \Longrightarrow mispred\ pstate\ [pcOf\ cfg]$
$\Longrightarrow$

(*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) $\to$*S* (*pstate*′, *cfg*′, *cfgs*′, *ibT*′, *ibUT*′, *ls*′)
$\longleftrightarrow$
($\exists$ *cfg1*′ *ibT1*′ *ibUT1*′. *pstate*′ = *update pstate* [*pcOf cfg*] $\wedge$
$\neg$ *finalM* (*cfg*, *ibT*, *ibUT*) $\wedge$ (*cfg*′, *ibT*′, *ibUT*′) = *nextB* (*cfg*, *ibT*, *ibUT*) $\wedge$
(*cfg1*′, *ibT1*′, *ibUT1*′) = *nextM* (*cfg*, *ibT*, *ibUT*) $\wedge$
cfgs′ = [*cfg1*′] $\wedge$ *ls*′ = *ls* $\cup$ *readLocs cfg*)
apply(*subst stepS.simps*) **by** *auto*

lemma *stepS-spec-normal-iff* [*simp*]:
cfgs $\neq$ [] $\Longrightarrow$
$\neg$ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*) $\Longrightarrow$
$\neg$ *is-IfJump* (*prog*!(*pcOf* (*last cfgs*))) $\vee$ $\neg$ *mispred pstate* (*pcOf cfg* # *map pcOf cfgs*) $\Longrightarrow$
prog!(*pcOf* (*last cfgs*)) $\neq$ *Fence*
$\Longrightarrow$
(*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) $\to$*S* (*pstate*′, *cfg*′, *cfgs*′, *ibT*′, *ibUT*′, *ls*′)
$\longleftrightarrow$
($\exists$ *cfg1*′. *pstate*′ = *pstate* $\wedge$
$\neg$ *is-getInput* (*prog*!(*pcOf* (*last cfgs*))) $\wedge$
$\neg$ *is-getInput* (*prog*!(*pcOf* (*last cfgs*))) $\wedge$ $\neg$ *is-Output* (*prog*!(*pcOf* (*last cfgs*))) $\wedge$
$\neg$ *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 $\neq$ [] $\Longrightarrow$
$\neg$ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*) $\Longrightarrow$
is-IfJump (*prog*!(*pcOf* (*last cfgs*))) $\Longrightarrow$ *mispred pstate* (*pcOf cfg* # *map pcOf cfgs*)
$\Longrightarrow$
(*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) $\to$*S* (*pstate*′, *cfg*′, *cfgs*′, *ibT*′, *ibUT*′, *ls*′)
$\longleftrightarrow$
($\exists$ *cfg1*′ *ibT1*′ *ibUT1*′ *lcfg*′. *pstate*′ = *update pstate* (*pcOf cfg* # *map pcOf cfgs*) $\wedge$

$\neg$ *finalM* (*last cfgs*, *ibT*, *ibUT*) $\wedge$
(*lcfg*′, *ibT*′, *ibUT*′) = *nextB* (*last cfgs*, *ibT*, *ibUT*) $\wedge$
(*cfg1*′, *ibT1*′, *ibUT1*′) = *nextM* (*last cfgs*, *ibT*, *ibUT*) $\wedge$
cfg′ = *cfg* $\wedge$ *cfgs*′ = *butlast cfgs* @ [*lcfg*′] @ [*cfg1*′] $\wedge$ *ls*′ = *ls* $\cup$ *readLocs* (*last cfgs*))
apply(*subst stepS.simps*) **by** *auto*

lemma *stepS-spec-Fence-iff* [*simp*]:
cfgs $\neq$ [] $\Longrightarrow$
$\neg$ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*) $\Longrightarrow$
prog!(*pcOf* (*last cfgs*)) = *Fence*
$\Longrightarrow$
(*pstate*, *cfg*, *cfgs*, *ibT*, *ibUT*, *ls*) $\to$*S* (*pstate*′, *cfg*′, *cfgs*′, *ibT*′, *ibUT*′, *ls*′)
$\longleftrightarrow$

$(pstate' = pstate \wedge cfg = cfg' \wedge cfgs' = [] \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' = ls)$
apply(*subst stepS.simps*) **by** *auto*

lemma *stepS-spec-resolve-iff*[*simp*]:
$cfgs \neq [] \Longrightarrow$
resolve pstate (*pcOf cfg* # *map pcOf cfgs*)
$\Longrightarrow$
$(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow S\ (pstate', cfg', cfgs', ibT', ibUT', ls')$
$\longleftrightarrow$
$(pstate' = update\ pstate\ (pcOf\ cfg\ \#\ map\ pcOf\ cfgs) \wedge$
$cfg' = cfg \wedge cfgs' = butlast\ cfgs \wedge ibT' = ibT \wedge ibUT' = ibUT \wedge ls' = ls)$
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]:
assumes $(pstate, cfg, cfgs, ibT, ibUT, ls) \rightarrow S\ (pstate', cfg', cfgs', ibT', ibUT', ls')$
obtains

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

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

$cfgs \neq []$
$\neg$ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*)
$\neg$ *is-IfJump* (*prog*!(*pcOf* (*last cfgs*))) $\vee \neg$ *mispred pstate* (*pcOf cfg* # *map pcOf cfgs*)
prog!(*pcOf* (*last cfgs*)) $\neq$ *Fence*
$pstate' = pstate$

¬ *is-getInput* (*prog*!(*pcOf* (*last cfgs*)))
¬ *is-Output* (*prog*!(*pcOf* (*last cfgs*)))
cfg′ = *cfg*
ls′ = *ls* ∪ *readLocs* (*last cfgs*)
∃ *cfg1′*. *nextB* (*last cfgs*, *ibT*, *ibUT*) = (*cfg1′*,*ibT′*,*ibUT′*)
∧ *cfgs′* = *butlast cfgs* @ [*cfg1′*]
|

cfgs ≠ []
¬ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*)
is-IfJump (*prog*!(*pcOf* (*last cfgs*))) *mispred pstate* (*pcOf cfg* # *map pcOf cfgs*)
pstate′ = *update pstate* (*pcOf cfg* # *map pcOf cfgs*)
¬ *finalM* (*last cfgs*, *ibT*, *ibUT*)
cfg′ = *cfg*
∃ *lcfg′ cfg1′ ibT1′ ibUT1′*.
nextB (*last cfgs*, *ibT*, *ibUT*) = (*lcfg′*,*ibT′*,*ibUT′*) ∧
(*cfg1′*, *ibT1′*, *ibUT1′*) = *nextM* (*last cfgs*, *ibT*, *ibUT*) ∧
cfgs′ = *butlast cfgs* @ [*lcfg′*] @ [*cfg1′*]
ls′ = *ls* ∪ *readLocs* (*last cfgs*)
|

cfgs ≠ []
¬ *resolve pstate* (*pcOf cfg* # *map pcOf cfgs*)
prog!(*pcOf* (*last cfgs*)) = *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*
using *assms* **by** (*cases rule*: *stepS.cases*, *metis*+)

lemma *stepS-endPC*: *pcOf cfg* = *endPC* ⟹ ¬ (*pstate*, *cfg*, [], *ibT*, *ibUT*, *ls*) →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-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′*, *cfgs′*, *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 ‹(*ps3*, *cfg3*, *cfgs3*, *ibT3*,*ibUT3*, *ls3*) →*S* (*ps3′*, *cfg3′*, *cfgs3′*, *ibT3′*,*ibUT3′*, *ls3′*)›
    and ‹(*ps4*, *cfg4*, *cfgs4*, *ibT4*,*ibUT4*, *ls4*) →*S* (*ps4′*, *cfg4′*, *cfgs4′*, *ibT4′*,*ibUT4′*, *ls4′*)›
      and ‹*cfg3* = (*Config pc3* (*State* (*Vstore vs3*) *avst3 h3 p3*))› **and** ‹*cfg3′* = (*Config pc3′* (*State* (*Vstore vs3′*) *avst3′ h3′ p3′*))›
      and ‹*cfg4* = (*Config pc4* (*State* (*Vstore vs4*) *avst4 h4 p4*))› **and** ‹*cfg4′* = (*Config pc4′* (*State* (*Vstore vs4′*) *avst4′ h4′ p4′*))›
     and ‹*cfgs3* = []› **and** ‹*cfgs4* = []›
     and ‹*prog*!*pc3* = (*x* ::= *a*)› **and** ‹*pcOf cfg3* = *pcOf cfg4*›
    shows ‹*cfgs3′* = [] ∧ *cfgs4′* = [] ∧
          vs3′ = (*vs3*(*x* := *aval a* (*stateOf cfg3*))) ∧
          vs4′ = (*vs4*(*x* := *aval a* (*stateOf cfg4*))) ∧
          pc3′ = *Suc pc3* ∧ *pc4′* = *Suc pc4* ∧ *ls4′* = *ls4* ∪ *readLocs cfg4* ∧
          avst3′ = *avst3* ∧ *avst4′* = *avst4* ∧ *ls3′* = *ls3* ∪ *readLocs cfg3* ∧
          p3 = *p3′* ∧ *p4* = *p4′*›
  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 instantiaton infrastructure for all our running examples. For a detailed explanation of each example and it's (dis)proof of Relative Security see the work by Dongol et al. [2]

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

no-notation *bot* (⊥)

abbreviation *noninform* (⊥) **where** ⊥ ≡ *undefined*

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

definition *noMisSpec* **where** *noMisSpec* (*cfgs*::*config list*) ≡ (*cfgs* = [])
lemma *noMisSpec-ext*[*simp*]:*map x cfgs* = *map x cfgs′* ⟹ *noMisSpec cfgs* ⟷ *noMisSpec cfgs′*
  by (*auto simp*: *noMisSpec-def*)

definition *misSpecL1* **where** *misSpecL1* (*cfgs*::*config list*) ≡ (*length cfgs* = *Suc 0*)
lemma *misSpecL1-len*[*simp*]:*misSpecL1 cfgs* ⟷ *length cfgs* = *1* **by** (*simp add*: *misSpecL1-def*)

definition *misSpecL2* **where** *misSpecL2* (*cfgs*::*config list*) ≡ (*length cfgs* = *2*)

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

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

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

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

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

context *Prog-Mispred*
begin

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

Vanilla-semantics system model: given by the normal semantics

type-synonym *stateV* = *config* × *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* ⇒ *bool*
begin

fun *istateV* :: *stateV* ⇒ *bool* **where**
istateV (*cfg*, *ibT*, *ibUT*, *ls*) ⟷
pcOf cfg = *0* ∧ *istate* (*stateOf cfg*) ∧
llength ibT = ∞ ∧ *llength ibUT* = ∞ ∧

ls = {}

fun *istateO* :: *stateO* ⇒ *bool* **where**
istateO (*pstate*, *cfg*, *cfgs*, *ibT*,*ibUT*, *ls*) ⟷
pstate = *initPstate* ∧
pcOf cfg = *0* ∧ *ls* = {} ∧
istate (*stateOf cfg*) ∧
cfgs = [] ∧ *llength ibT* = ∞ ∧ *llength ibUT* = ∞

lemma *istateV-config-imp*:
istateV (*cfg*, *ibT*,*ibUT*, *ls*) ⟹ *pcOf cfg* = *0* ∧ *ls* = {} ∧ *ibT* ≠ *LNil*
by *force*

lemma *istateO-config-imp*:
istateO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟹
cfgs = [] ∧ *pcOf cfg* = *0* ∧ *ls* = {} ∧ *ibT* ≠ *LNil*
unfolding *istateO.simps*
by *auto*

definition *same-var-all x cfg1 cfg2 cfg3 cfgs3 cfg4 cfgs4* ≡
vstore (*getVstore* (*stateOf cfg1*)) *x* = *vstore* (*getVstore* (*stateOf cfg4*)) *x* ∧
vstore (*getVstore* (*stateOf cfg2*)) *x* = *vstore* (*getVstore* (*stateOf cfg4*)) *x* ∧
vstore (*getVstore* (*stateOf cfg3*)) *x* = *vstore* (*getVstore* (*stateOf cfg4*)) *x* ∧
(∀ *cfg3*′∈*set cfgs3*. *vstore* (*getVstore* (*stateOf cfg3*′)) *x* = *vstore* (*getVstore* (*stateOf cfg3*)) *x*) ∧
(∀ *cfg4*′∈*set cfgs4*. *vstore* (*getVstore* (*stateOf cfg4*′)) *x* = *vstore* (*getVstore* (*stateOf cfg4*)) *x*)

definition *same-var x cfg cfg*′ ≡
vstore (*getVstore* (*stateOf cfg*)) *x* = *vstore* (*getVstore* (*stateOf cfg*′)) *x*

definition *same-var-val x* (*val*::*int*) *cfg cfg*′ ≡
vstore (*getVstore* (*stateOf cfg*)) *x* = *vstore* (*getVstore* (*stateOf cfg*′)) *x* ∧
vstore (*getVstore* (*stateOf cfg*)) *x* = *val*

definition *same-var-o ii cfg3 cfgs3 cfg4 cfgs4* ≡
vstore (*getVstore* (*stateOf cfg3*)) *ii* = *vstore* (*getVstore* (*stateOf cfg4*)) *ii* ∧
(∀ *cfg3*′∈*set cfgs3*. *vstore* (*getVstore* (*stateOf cfg3*′)) *ii* = *vstore* (*getVstore* (*stateOf cfg3*)) *ii*) ∧
(∀ *cfg4*′∈*set cfgs4*. *vstore* (*getVstore* (*stateOf cfg4*′)) *ii* = *vstore* (*getVstore* (*stateOf*

cfg4)) *ii*)

lemma *set-var-shrink*:∀ *cfg3'*∈*set cfgs*.
vstore (*getVstore* (*stateOf cfg3'*)) *var* =
vstore (*getVstore* (*stateOf cfg*)) *var*
⟹
∀ *cfg3'*∈*set* (*butlast cfgs*).
vstore (*getVstore* (*stateOf cfg3'*)) *var* =
vstore (*getVstore* (*stateOf cfg*)) *var*
by (*meson in-set-butlastD*)

lemma *heapSimp*:(∀ *cfg''*∈*set cfgs''*. *getHheap* (*stateOf cfg'*) = *getHheap* (*stateOf cfg''*)) ∧ *cfgs''* ≠ []
⟹ *getHheap* (*stateOf cfg'*) = *getHheap* (*stateOf* (*last cfgs''*))
by *simp*

lemma *heapSimp2*:(∀ *cfg''*∈*set cfgs''*. *getHheap* (*stateOf cfg'*) = *getHheap* (*stateOf cfg''*)) ∧ *cfgs''* ≠ []

⟹ *getHheap* (*stateOf cfg'*) = *getHheap* (*stateOf* (*hd cfgs''*))
by *simp*

lemma *array-baseSimp*:*array-base aa1* (*getAvstore* (*stateOf cfg*)) =
array-base aa1 (*getAvstore* (*stateOf cfg'*)) ∧
(∀ *cfg'*∈*set cfgs*. *array-base aa1* (*getAvstore* (*stateOf cfg'*)) =
array-base aa1 (*getAvstore* (*stateOf cfg*)))
∧ *cfgs* ≠ []
⟹
array-base aa1 (*getAvstore* (*stateOf cfg*)) =
array-base aa1 (*getAvstore* (*stateOf* (*last cfgs*)))

by *simp*

lemma *finalB-imp-finalS*:*finalB* (*cfg*, *ibT*,*ibUT*) ⟹ (∀ *pstate cfgs ls*. *finalS* (*pstate*, *cfg*, [], *ibT*,*ibUT*, *ls*))
unfolding *finalB-def finalS-def final-def* **apply** *clarsimp*
subgoal for *pstate ls pstate' cfg' cfgs' 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*]:*length cfgs* = *Suc 0* ⟹ *cfgs* = [*last cfgs*]
by (*metis Suc-length-conv last-ConsL length-0-conv*)

lemma *cfgs-map*[*simp*]:*length cfgs* = *Suc 0* ⟹ *map pcOf cfgs* = [*pcOf* (*last 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*))

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* (‹⊥›)
type-synonym *secret* = *state*

context *Prog-Mispred*
begin

fun *corrState* :: *stateV* ⇒ *stateO* ⇒ *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* ⇒ *bool* **where**
isSecV ss ≡ *case ss of* (*cfg*,*ibT*,*ibUT*) ⇒ (*pcOf cfg* = *0*)

We consider the entire initial state as a secret:

fun *getSecV* :: *stateV* ⇒ *secret* **where**
getSecV (*cfg*,*ibT*,*ibUT*) = *stateOf cfg*

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

definition *isSecO* :: *stateO* ⇒ *bool* **where**
isSecO ss ≡ *case ss of* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⇒ (*pcOf cfg* = *0* ∧ *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*]:
(*prog*!(*pcOf cfg*)) = *Input T s* ⟹ *getActV* (*cfg*,*ibT*,*ibUT*,*ls*) = *lhd ibT*
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getAct-def*)

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

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

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

lemma *not-is-Output-getObsV*[*simp*]:
¬ *is-Output* (*prog*!(*pcOf cfg*)) ⟹ *getObsV* (*cfg*,*ibT*,*ibUT*,*ls*) = ⊥
apply (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getObs-def*)
subgoal for *x* **by** (*cases x*, *simp-all*) **.**

lemma *is-getTrustedInput-Nil-getActO*[*simp*]:
(*prog*!(*pcOf cfg*)) = *Input T s* ⟹ *getActO* (*pstate*,*cfg*,[],*ibT*,*ibUT*,*ls*) = *lhd ibT*
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Opt.getAct-def*)

lemma *not-is-getTrustedInput-Nil-getActO*[*simp*]:
¬ *is-getInput* (*prog*!(*pcOf cfg*))
∨ *cfgs* ≠ [] ⟹ *getActO* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) = ⊥
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*]:
prog!(*pcOf cfg*) = *Output U s* ⟹
getObsO (*pstate*,*cfg*,[],*ibT*,*ibUT*,*ls*) = (*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Opt.getObs-def*)

lemma *not-is-Output-Nil-getObsO*[*simp*]:
¬ *is-Output* (*prog*!(*pcOf cfg*)) ∨ *cfgs* ≠ [] ⟹ *getObsO* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*)
= ⊥
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*:
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*case prog*!(*pcOf cfg*) *of*
Input T - ⇒ *lhd ibT*
|*Input U* - ⇒ *lhd ibUT*
|- ⇒ ⊥
)
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 - ⇒ (*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
|- ⇒ ⊥
)
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* -) ⇒ *lhd ibT*
|([],*Input U* -) ⇒ *lhd ibUT*
|- ⇒ ⊥
)
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* -) ⇒ (*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
|- ⇒ ⊥
)
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* (‹⊥›)

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*: *ffileVal-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*, ⊥, ⊥)
   |*Input T* - ⇒ (⊥, *lhd ibT*, ⊥)
   |*Output T* - ⇒ (⊥,⊥,*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*))
   |- ⇒ (⊥,⊥,⊥))

lemma *isSecV-iff*:*isSecV ss* ⟷ ¬*finalN ss*
  unfolding *isSecV-def*
  by (*simp add*: *case-prod-beta*)

definition *isSecO* :: *stateO* ⇒ *bool* **where**
isSecO ss ≡ *case ss of* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⇒ ¬*finalS ss* ∧ *cfgs* = []
fun *getSecO* :: *stateO* ⇒ *secret* **where**
getSecO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
 (*case prog*!(*pcOf cfg*) *of*

Start ⇒ (*stateOf cfg*, ⊥, ⊥)
|*Input T* - ⇒ (⊥, *lhd ibT*, ⊥)
|*Output T* - ⇒ (⊥,⊥,*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*))
|- ⇒ (⊥,⊥,⊥))
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*]:
(*prog*!(*pcOf cfg*)) = *Input U inp* ⟹ *getActV* (*cfg*,*ibT*,*ibUT*,*ls*) = *lhd ibUT*
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getAct-def*)

lemma *is-getInputU-getActV*[*simp*]:
(*prog*!(*pcOf cfg*)) = *Input T inp* ⟹ *getActV* (*cfg*,*ibT*,*ibUT*,*ls*) = *lhd ibT*
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getAct-def*)

lemma *not-is-getInput-getActV*[*simp*]:
¬ *is-getInput* (*prog*!(*pcOf cfg*)) ⟹ *getActV* (*cfg*,*ibT*,*ibUT*,*ls*) = ⊥
apply (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getAct-def*)
  subgoal for *t* **apply**(*cases t*, *simp-all*) **. .**

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

lemma *not-is-Output-getObsV*[*simp*]:
¬ *is-Output* (*prog*!(*pcOf cfg*)) ⟹ *getObsV* (*cfg*,*ibT*,*ibUT*,*ls*) = ⊥
apply (*cases prog*!(*pcOf cfg*), *auto simp*: *Van.getObs-def*)
  subgoal for *t* **apply**(*cases t*, *simp-all*) **. .**

lemma *is-getInputT-Nil-getActO*[*simp*]:
(*prog*!(*pcOf cfg*)) = *Input T inp* ⟹ *getActO* (*pstate*,*cfg*,[],*ibT*,*ibUT*,*ls*) = *lhd ibT*
  by (*cases prog*!(*pcOf cfg*), *auto simp*: *Opt.getAct-def*)

lemma *is-getInputU-Nil-getActO*[*simp*]:
(*prog*!(*pcOf cfg*)) = *Input U inp* ⟹ *getActO* (*pstate*,*cfg*,[],*ibT*,*ibUT*,*ls*) = *lhd ibUT*
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Opt.getAct-def*)

lemma *not-is-getInput-Nil-getActO*[*simp*]:
(¬ *is-getInput* (*prog*!(*pcOf cfg*)))
∨ *cfgs* ≠ [] ⟹ *getActO* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) = ⊥
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*]:
(*prog*!(*pcOf cfg*)) = *Output U out* ⟹
 getObsO (*pstate*,*cfg*,[],*ibT*,*ibUT*,*ls*) = (*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
by (*cases prog*!(*pcOf cfg*), *auto simp*: *Opt.getObs-def*)

lemma *not-is-Output-Nil-getObsO*[*simp*]:
¬ *is-Output* (*prog*!(*pcOf cfg*)) ∨ *cfgs* ≠ [] ⟹ *getObsO* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*)
= ⊥
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 -   ⇒ lhd ibT
   |Input U - ⇒ lhd ibUT
   |- ⇒ ⊥
 )
  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 - ⇒ (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
   |- ⇒ ⊥
 )
  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 -)   ⇒ lhd ibT
   |([],Input U -) ⇒ lhd ibUT
   |- ⇒ ⊥
 )
  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 -) ⇒ (outOf (prog!(pcOf cfg)) (stateOf cfg), ls)
   |- ⇒ ⊥
 )
  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* (‹⊥›)
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-add* = {*a*. *a* ≠ *nat NN*+*1*}
fun vs_0::*char list* ⇒ *int* **where**
vs_0 *x* = *0*

lemma *vs0*[*simp*]:(λ*x*. *0*) = vs_0 **unfolding** vs_0*.simps* **by** *simp*

fun *as*:: *char list* ⇒ *nat* × *nat* **where**
as a = (*if a* = *aa1 then* (*0*, *nat NN*)
else (*if a* = *aa2 then* (*nat NN*, *nat size-aa2*)
else (*nat size-aa2*,*0*)))

definition *avst′* ≡ (*Avstore as*)

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

lemma *avstore-loc*[*simp*]:*Avstore* (λ*a. if a* = *aa1 then* (*0*, *nat NN*) *else if a* = *aa2 then* (*nat NN*, *nat size-aa2*) *else* (*nat size-aa2*, *0*)) =
avst′
unfolding *avst-defs* **by** *auto*

abbreviation *read-add* ≡ {*a. a* ≠ (*nat NN* + *1*)}

fun *initVstore* :: *vstore* ⇒ *bool* **where**
initVstore (*Vstore vst*) = (*vst* = vs_0)

fun *initAvstore* :: *avstore* ⇒ *bool* **where**
initAvstore avst = (*avst* = *avst′*)
fun *initHeap*::(*nat* ⇒ *int*) ⇒ *bool* **where**
initHeap h = (∀ *x*∈*read-add. h x* = *0*)

lemma *initAvstore-0*[*intro*]:*initAvstore avst′* ⟹ *array-base aa1 avst′* = *0*
unfolding *avst-defs array-base-def*
by (*smt* (*verit*, *del-insts*) *avstore.case fstI*)

fun *istate* ::*state* ⇒ *bool* **where**
istate s =
(*initVstore* (*getVstore s*) ∧
initAvstore (*getAvstore s*) ∧
initHeap (*getHheap s*))

definition *prog* ≡
[
0/ *Start* ,
1/ *Input U xx* ,
2/ *tt* ::= (*N 0*),
3/ *IfJump* (*Less* (*V xx*) (*N NN*)) *4 5*,
4/ *tt* ::= (*VA aa2* (*Times* (*VA aa1* (*V xx*)) (*N 512*))) ,
5/ *Output U* (*V tt*)
]

lemma *cases-5*: (*i*::*pcounter*) = *0* ∨ *i* = *1* ∨ *i* = *2* ∨ *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 *xx-NN-cases*: *vs xx* < (*int NN*) ∨ *vs xx* ≥ (*int NN*) **by** *auto*

lemma *is-If-pcOf*[*simp*]:
pcOf cfg < *6* ⟹ *is-IfJump* (*prog* ! (*pcOf cfg*)) ⟷ *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* ⟹ *is-IfJump* (*prog* ! *pc*) ⟷ *pc* = *3*
using *cases-5*[*of pc*]
by (*auto simp*: *prog-def*)

lemma *eq-Fence-pc*[*simp*]:
pc < *6* ⟹ *prog* ! *pc* ≠ *Fence*
using *cases-5*[*of pc*]
by (*auto simp*: *prog-def*)

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

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

consts *update* :: *predState* ⇒ *pcounter list* ⇒ *predState*
consts $pstate_0$ :: *predState*

interpretation *Prog-Mispred-Init* **where**
prog = *prog* **and** *initPstate* = $pstate_0$ **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*]: (*prog*!*1*) = *Input U xx*
by (*auto simp*: *prog-def*)

lemma *is-Output-pcOf*[*simp*]: *pcOf cfg* < *6* ⟹ *is-Output* (*prog*!(*pcOf cfg*)) ⟷ *pcOf cfg* = *5* ∨ *pcOf cfg* = *6*
using *cases-5*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *is-Fence-pcOf*[*simp*]: *pcOf cfg* < *6* ⟹(*prog*!(*pcOf cfg*)) ≠ *Fence*
using *cases-5*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *prog0* [*simp*]:*prog* ! *0* = *Start*
by (*auto simp*: *prog-def*)

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

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

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

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

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

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

lemma *isSecO-pcOf* [*simp*]:
isSecO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟷ (*pcOf cfg* = *0* ∧ *cfgs* = [])
using *isSecO-def* **by** *simp*

lemma *getInputT-not*[*simp*]: *pcOf cfg* < *6* ⟹
(*prog* ! *pcOf cfg*) ≠ *Input T x*
apply(*cases cfg*) **subgoal for** *pc s* **using** *cases-5* [*of pcOf cfg*]
by (*auto simp*: *prog-def*) **.**

lemma *getActV-pcOf* [*simp*]:
pcOf cfg < *6* ⟹
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1 then lhd ibUT else* ⊥)
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* ⟹
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *5 then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥

)
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* ⟹
getActO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1* ∧ *cfgs* = [] *then lhd ibUT else* ⊥)
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* ⟹
getObsO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if* (*pcOf cfg* = *5* ∧ *cfgs* = []) *then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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 ≠ *LNil* ⟹ *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 ≠ *LNil* ⟹ *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* ⟹
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 ≥ *NN* ⟹
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*:
$pc < 6 \Longrightarrow (pc = 1 \longrightarrow ibUT \neq LNil) \Longrightarrow$
$(Config\ pc\ s,\ ibT,\ ibUT) \rightarrow B\ nextB\ (Config\ pc\ s,\ ibT,\ ibUT)$
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*:
$pc < 6 \Longrightarrow (pc = 1 \longrightarrow ibUT \neq LNil) \Longrightarrow$
$\neg$ *finalB* (*Config pc s*, *ibT*, *ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff*′:
$pc < 6 \Longrightarrow$
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* **by** *auto* **. .**

lemma *finalB-pc-iff*:
$pc \leq 6 \Longrightarrow$
finalB (*Config pc s*, *ibT*, *ibUT*) $\longleftrightarrow$
($pc = 1 \wedge ibUT = LNil \vee pc = 6$)
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 cfg $\leq 6 \Longrightarrow$
finalB (*cfg*, *ibT*, *ibUT*) $\longleftrightarrow$ (*pcOf cfg* $= 1 \wedge ibUT = LNil \vee$ *pcOf cfg* $= 6$)
by (*metis config.collapse finalB-pc-iff*)

definition vs_i*-t cfg* $\equiv$ (*vstore* (*getVstore* (*stateOf cfg*)) *xx*) $<$ *NN*
definition vs_i*-f cfg* $\equiv$ (*vstore* (*getVstore* (*stateOf cfg*)) *xx*) $\geq$ *NN*
lemma *vs-xx-cases*:vs_i*-t cfg* $\vee$ vs_i*-f cfg* **unfolding** vs_i*-t-def* vs_i*-f-def* **by** *auto*

lemmas vs_i*-defs* = vs_i*-t-def* vs_i*-f-def*

lemma *bool-invar*[*simp*]:$\neg vs_i$*-t* (*Config 6 s*) $\Longrightarrow$ vs_i*-t* (*Config 6 s*) $\Longrightarrow$ (*Config 6 s*, *ib1*) $\rightarrow B$ (*Config 6 s*, *ib1*) $\Longrightarrow$ *False*
unfolding vs_i*-defs*
by *simp*
lemma *nextB-vs-consistent-aux*:
$2 \leq pc \wedge pc < 6 \Longrightarrow$
(*nextB* (*Config pc* (*State* (*Vstore vs*) *avst* (*Heap h*) *p*),*ibT*, *ibUT*)) = (*Config pc′* (*State* (*Vstore vs′*) *avst′′* (*Heap h′*) *p′*),*ibT′*, *ibUT′*) $\Longrightarrow$
avst = *avst′′* $\wedge$
vs xx = *vs′ 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 cfg* $\wedge$ *pcOf cfg* $< 6 \Longrightarrow$
(*nextB* (*cfg*, *ibT*, *ibUT*)) = (*cfg′*, *ibT′*, *ibUT′*) $\Longrightarrow$
(*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-vs-consistent-aux* **apply** *simp*
by *blast* **. . . . . .**

lemma *nextB-vs*$_i$*-t-consistent*:
$2 \leq pcOf\ cfg \wedge pcOf\ cfg < 6 \Longrightarrow$
$(nextB\ (cfg,\ ibT,\ ibUT)) = (cfg',\ ibT',\ ibUT') \Longrightarrow$
$vs_i\text{-}t\ cfg \longleftrightarrow vs_i\text{-}t\ cfg'$
unfolding *vs*$_i$*-defs* **using** *nextB-vs-consistent*
by *simp*

lemma *nextB-vs*$_i$*-f-consistent*:
$2 \leq pcOf\ cfg \wedge pcOf\ cfg < 6 \Longrightarrow$
$(nextB\ (cfg,\ ibT,\ ibUT)) = (cfg',\ ibT',\ ibUT') \Longrightarrow$
$vs_i\text{-}f\ cfg \longleftrightarrow vs_i\text{-}f\ cfg'$
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 cfg3*)) *xx* = *vstore* (*getVstore* (*stateOf cfg4*)) *xx* $\wedge$
($\forall$ *cfg3'*$\in$*set cfgs3*. *vstore* (*getVstore* (*stateOf cfg3'*)) *xx* = *vstore* (*getVstore* (*stateOf cfg3*)) *xx*) $\wedge$
($\forall$ *cfg4'*$\in$*set cfgs4*. *vstore* (*getVstore* (*stateOf cfg4'*)) *xx* = *vstore* (*getVstore* (*stateOf cfg4*)) *xx*)

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

definition $pstate_1 \equiv update\ pstate_0\ [3]$
definition $pstate_2 \equiv update\ pstate_1\ [5,5]$

lemmas *pstate-def* = *pstate*$_1$*-def pstate*$_2$*-def*

fun hh_3:: *nat* ⇒ *int* **where**
$hh_3\ x = \ (\mathit{if}\ x = (\mathit{nat}\ NN + 1)\ \mathit{then}\ 5\ \mathit{else}\ 0)$

definition $h_3 \equiv (\mathit{Heap}\ hh_3)$

fun hh_4:: *nat* ⇒ *int* **where**
$hh_4\ x = (\mathit{if}\ x = (\mathit{nat}\ NN + 1)\ \mathit{then}\ 6\ \mathit{else}\ 0)$

definition $h_4 \equiv (\mathit{Heap}\ hh_4)$

lemmas *h-def* = h_3-*def* h_4-*def* hh_3.*simps* hh_4.*simps*

lemma *ss-neq-aux1*:$\mathit{nat}(5 * 512) \neq \mathit{nat}\ (6 * 512)$ **by** *auto*
lemma *ss-neq-aux2*:$\mathit{nat}(3 * 512) \neq \mathit{nat}\ (5 * 512)$ **by** *auto*
lemmas *ss-neq* = *ss-neq-aux1 ss-neq-aux2*

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

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

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

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

definition $aa2_{vs3} \equiv$ *array-loc aa2* $(\mathit{nat}\ (hh_3\ aa1_i * 512))\ avst'$

definition $vs_{33} = vs_2(tt := hh_3\ aa2_{vs3})$

definition $aa2_{vs4} \equiv$ *array-loc aa2* $(\mathit{nat}\ (hh_4\ aa1_i * 512))\ avst'$

definition $vs_{34} = vs_2(tt := hh_4\ aa2_{vs4})$

lemmas $reads_m$-*def* = $aa1_i$-*def* $aa2_{vs3}$-*def* $aa2_{vs4}$-*def*
lemmas *vs-def* = vs_0.*simps* vs_1-*def* vs_2-*def* vs_{33}-*def* vs_{34}-*def*

definition $s_{03} \equiv (\mathit{State}\ (\mathit{Vstore}\ vs_0)\ avst'\ h_3\ p)$
definition $s_{13} \equiv (\mathit{State}\ (\mathit{Vstore}\ vs_1)\ avst'\ h_3\ p)$

definition $s_{23} \equiv (State\ (Vstore\ vs_2)\ avst'\ h_3\ p)$
definition $s_{33} \equiv (State\ (Vstore\ vs_{33})\ avst'\ h_3\ p)$

definition $s_{04} \equiv (State\ (Vstore\ vs_0)\ avst'\ h_4\ p)$
definition $s_{14} \equiv (State\ (Vstore\ vs_1)\ avst'\ h_4\ p)$
definition $s_{24} \equiv (State\ (Vstore\ vs_2)\ avst'\ h_4\ p)$
definition $s_{34} \equiv (State\ (Vstore\ vs_{34})\ avst'\ h_4\ p)$

lemmas *s-def* = s_{03}*-def* s_{13}*-def* s_{23}*-def* s_{33}*-def*
s_{04}*-def* s_{14}*-def* s_{24}*-def* s_{34}*-def*

definition $(s3_0{::}\ stateO) \equiv (pstate_0,\ (Config\ 0\ s_{03}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{\})$
definition $(s3_1{::}\ stateO) \equiv (pstate_0,\ (Config\ 1\ s_{03}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{\})$
definition $(s3_2{::}\ stateO) \equiv (pstate_0,\ (Config\ 2\ s_{13}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{\})$
definition $(s3_3{::}\ stateO) \equiv (pstate_0,\ (Config\ 3\ s_{23}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{\})$
definition $(s3_4{::}\ stateO) \equiv (pstate_1,\ (Config\ 5\ s_{23}),\ [Config\ 4\ s_{23}],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{\})$
definition $(s3_5{::}\ stateO) \equiv (pstate_1,\ (Config\ 5\ s_{23}),\ [Config\ 5\ s_{33}],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{aa2_{vs3},\ aa1_i\})$
definition $(s3_6{::}\ stateO) \equiv (pstate_2,\ (Config\ 5\ s_{23}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{aa2_{vs3},\ aa1_i\})$
definition $(s3_7{::}\ stateO) \equiv (pstate_2,\ (Config\ 6\ s_{23}),\ [],\ repeat\ (NN{+}1),\ repeat\ (NN{+}1),\ \{aa2_{vs3},\ aa1_i\})$

lemmas *s3-def* = $s3_0$*-def* $s3_1$*-def* $s3_2$*-def* $s3_3$*-def* $s3_4$*-def* $s3_5$*-def* $s3_6$*-def* $s3_7$*-def*

lemmas *state-def* = *s-def h-def vs-def* $reads_m$*-def pstate-def avst-defs*

definition *s3-trans* $\equiv [s3_0,\ s3_1,\ s3_2,\ s3_3,\ s3_4,\ s3_5,\ s3_6,\ s3_7]$
lemmas *s3-trans-defs* = *s3-trans-def s3-def*

lemma *hd-s3-trans*[*simp*]: *hd s3-trans* = $s3_0$ **by** (*simp add*: *s3-trans-def*)
lemma *s3-trans-nemp*[*simp*]: *s3-trans* ≠ [] **by** (*simp add*: *s3-trans-def*)

lemma $s3_{01}$[*simp*]:$s3_0 \to S\ s3_1$
  unfolding *s3-def*
  using *nonspec-normal*
  by *simp*

lemma $s3_{12}$[*simp*]:$s3_1 \to S\ s3_2$

unfolding *s3-def state-def*
using *nonspec-normal*
by *simp*

lemma $s3_{23}$[*simp*]:$s3_2 \rightarrow S\ s3_3$
unfolding *s3-def state-def*
by (*simp add*: *finalM-iff*)

lemma $s3_{34}$[*simp*]:$s3_3 \rightarrow S\ s3_4$
unfolding *s3-def state-def*
using *nonspec-mispred*
by (*simp add*: *finalM-iff*)

lemma $s3_{45}$[*simp*]:$s3_4 \rightarrow S\ s3_5$
unfolding *s3-def state-def*
using *spec-normal*
by (*simp-all add*: *finalM-iff*, *blast*)

lemma $s3_{56}$[*simp*]:$s3_5 \rightarrow S\ s3_6$
unfolding *s3-def state-def*
using *spec-resolve*
by *simp*

lemma $s3_{67}$[*simp*]:$s3_6 \rightarrow S\ s3_7$
unfolding *s3-def state-def*
using *nonspec-normal*
by *simp*

lemma *finalS-*$s3_7$[*simp*]:*finalS* $s3_7$
unfolding *finalS-def final-def s3-def*
by (*simp add*: *stepS-endPC*)

lemmas *s3-trans-simps* = $s3_{01}$ $s3_{12}$ $s3_{23}$ $s3_{34}$ $s3_{45}$ $s3_{56}$ $s3_{67}$

definition ($s4_0$:: *stateO*) ≡ ($pstate_0$, (*Config 0* s_{04}), [], *repeat* (*NN+1*), *repeat* (*NN+1*), {})
definition ($s4_1$:: *stateO*) ≡ ($pstate_0$, (*Config 1* s_{04}), [], *repeat* (*NN+1*), *repeat* (*NN+1*), {})
definition ($s4_2$:: *stateO*) ≡ ($pstate_0$, (*Config 2* s_{14}), [], *repeat* (*NN+1*), *repeat* (*NN+1*), {})
definition ($s4_3$:: *stateO*) ≡ ($pstate_0$, (*Config 3* s_{24}), [], *repeat* (*NN+1*), *repeat* (*NN+1*), {})
definition ($s4_4$:: *stateO*) ≡ ($pstate_1$, (*Config 5* s_{24}), [*Config 4* s_{24}], *repeat* (*NN+1*), *repeat* (*NN+1*), {})
definition ($s4_5$:: *stateO*) ≡ ($pstate_1$, (*Config 5* s_{24}), [*Config 5* s_{34}], *repeat* (*NN+1*), *repeat* (*NN+1*), {$aa2_{vs4}$, $aa1_i$})
definition ($s4_6$:: *stateO*) ≡ ($pstate_2$, (*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* ≠ [] **by** (*simp add*: *s4-trans-def*)

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

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

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

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

lemma $s4_{45}$[*simp*]:$s4_4 \to S\ s4_5$
 unfolding *s4-def state-def*
 using *spec-normal*
 by (*simp add*: *finalM-iff*, *blast*)

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

lemma $s4_{67}$[*simp*]:$s4_6 \to S\ s4_7$
 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* = [s_{03}]
apply (*simp add*: *Opt.S-def filtermap-def*)
unfolding *s3-trans-defs* **by** *simp*

lemma *S-s4-trans*[*simp*]:*Opt.S s4-trans* = [s_{04}]
apply (*simp add*: *Opt.S-def filtermap-def*)
unfolding *s4-trans-defs* **by** *simp*

lemma *finalB-noStep*[*simp*]:$\bigwedge$*s1'*. *finalB* (*cfg1*, *ibT1*, *ibUT1*) $\Longrightarrow$ (*cfg1*, *ibT1*, *ibUT1*, *ls1*) →*N s1'* $\Longrightarrow$ *False*
unfolding *finalN-def final-def finalB-eq-finalN* **by** *auto*

8.2.3 Disproof of fun1

fun *common-memory*::*config* ⇒ *config* ⇒ *bool* **where**
common-memory cfg1 cfg2 =
(*let h1* = (*getHheap* (*stateOf cfg1*));
  h2 = (*getHheap* (*stateOf cfg2*)) *in*
((∀ *x*∈*read-add*. *h1 x* = *h2 x* ∧ *h1 x* = *0*) ∧
(*getAvstore* (*stateOf cfg1*)) = *avst'* ∧
(*getAvstore* (*stateOf cfg2*)) = *avst'*))

lemma *heap-eq0*[*simp*]: ∀ *x*. *x* ≠ *Suc NN* ⟶ *hh1' x* = *hh2' x* ∧ *hh1' x* = *0* ⟹
hh2' NN = *0*
 by (*metis n-not-Suc-n*)
lemma *heap1-eq0*[*simp*]:∀ *x*. *x* ≠ *Suc NN* ⟶ *hh1' x* = *hh2' x* ∧ *hh1' x* = *0* ⟹
  vs2 xx < *NN* ⟹
  hh2' (*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* ⇒ *state list* ⇒ *stateV* ⇒ *state list* ⇒ *bool* **where**
Γ-*inv* (*cfg1*,*ibT1*,*ibUT1*,*ls1*) *sl1* (*cfg2*,*ibT2*,*ibUT2*,*ls2*) *sl2* =
(
(*pcOf cfg1* = *pcOf cfg2*) ∧

(*pcOf cfg1* < *2* ⟶ *ibUT1* ≠ *LNil* ∧ *ibUT2* ≠ *LNil*) ∧

(*pcOf cfg1* > *2* ⟶ *same-var-val tt 0 cfg1 cfg2*) ∧

(*pcOf cfg1* > *1* ⟶ (*same-var xx cfg1 cfg2*) ∧

  (vs_i-*t cfg1* ⟶ *pcOf cfg1* ∈ *trueProg*) ∧

  (vs_i-*f cfg1* ⟶ *pcOf cfg1* ∈ *falseProg*))
∧
ls1 = *ls2* ∧

pcOf cfg1 ∈ *PC* ∧
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* ⟹

 pcOf cfg1 ≤ *6* ∧ *pcOf cfg2* ≤ *6* ∧

 (*pcOf cfg1* = *4* ⟶ vs_i-*t cfg1*) ∧

(*pcOf cfg2* = *4* ⟶ vs_i-*t cfg2*) ∧
(*pcOf cfg1* > *1* ⟶ vs_i-*t cfg1* ⟷ vs_i-*t cfg2*) ∧

(*finalB* (*cfg1*,*ibT1*,*ibUT1*) ⟷ *pcOf cfg1* = *6*) ∧
(*finalB* (*cfg2*,*ibT2*,*ibUT2*) ⟷ *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* $s3_0$ **unfolding** *s3-def state-def* **by** *simp*
lemma *istateO-s4*[*simp*]:*istateO* $s4_0$ **unfolding** *s4-def state-def* **by** *simp*

lemma *validFromS-s3*[*simp*]:*Opt.validFromS* $s3_0$ *s3-trans*
unfolding *Opt.validFromS-def* **by** *simp*

lemma *validFromS-s4*[*simp*]:*Opt.validFromS* $s4_0$ *s4-trans*
unfolding *Opt.validFromS-def* **by** *simp*

lemma *completedFromO-s3*[*simp*]:*completedFromO* $s3_0$ *s3-trans*
unfolding *Opt.completedFrom-def* **by** *simp*

lemma *completedFromO-s4*[*simp*]:*completedFromO* $s4_0$ *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*:$aa2_{vs3} \neq aa2_{vs4}$
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*:$aa2_{vs3} \neq aa1_i$
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*:$aa2_{vs4} \neq aa1_i$
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*. *istateV s1* ⟹ *corrState s1* $s3_0$ ⟹ *istateV s2* ⟹ *corrState s2* $s4_0$ ⟹ Γ-*inv s1* [s_{03}] *s2* [s_{04}]
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 getSecV 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 Γ **unfolding** *isIntV.simps ss*
unfolding Γ-*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 ¬ *isIntV ss1* ⟶ *move1* Γ-*inv ss1 sl1 ss2 sl2* ∧ *move2* Γ-*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* ⟶ *getActV ss1* = *getActV ss2* ⟶ *getObsV ss1* = *getObsV ss2* ∧ *move12* Γ-*inv ss1 sl1 ss2 sl2*
proof(*unfold ss isIntV.simps finalN-iff-finalB*, *intro impI*, *rule conjI*)
assume *final*:¬ *finalB* (*cfg1*, *ibT1*, *ibUT1*) **and**
getAct:*getActV* (*cfg1*, *ibT1*, *ibUT1*, *ls1*) = *getActV* (*cfg2*, *ibT2*, *ibUT2*, *ls2*)
have *not6*:*pc1* = *6* ⟹ *False*
using *cfg final* Γ
by *simp*

show *getObsV* (*cfg1*, *ibT1*, *ibUT1*, *ls1*) = *getObsV* (*cfg2*, *ibT2*, *ibUT2*, *ls2*)
using Γ *getAct* **unfolding** *ss*
apply−**apply**(*frule* Γ-*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* ∪ *readLocs cfg2* **using** *v* **unfolding** *ss* **by** *simp*
note *ls* = *ls1' ls2'*

note Γ-*simps* = *cfg ls* vs_i*-defs hh array-loc-def*
array-base-def state-def PC-def

show Γ-*inv ss1' sl1' ss2' sl2'*
using Γ *valid getAct*
unfolding *ss* **apply**−**apply**(*frule* Γ-*implies*)
using *cases-5*[*of pc1*] *not6* **apply**(*elim disjE*, *simp-all*)
unfolding Γ-*def ss*
prefer *4* **subgoal using** *vs-xx-cases*[*of cfg1*]

by (*elim disjE*, *unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)
subgoal by (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)
subgoal by (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)
subgoal by (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)
subgoal using *val-neq-1* **apply** (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)

using *val-neq-1* **by** (*metis NN-suc add-left-cancel nat-int*)
subgoal by (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*)
subgoal by (*unfold* Γ-*defs*, *auto simp add*: Γ-*simps*) **.**
qed
qed
qed

theorem ¬*rsecure*
apply(*rule unwindSD-rsecure*[*of* $s3_0$ *s3-trans* $s4_0$ *s4-trans* Γ-*inv*])
by *simp-all*

end

9 Proof of Relative Security for fun2

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

9.1 Function definition and Boilerplate

no-notation *bot* (‹⊥›)

consts *NN* :: *nat*
lemma *NN*: *NN* ≥ *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* $\geq$ *0* **and** *aa2*:*size-aa2* $\geq$ *0* **by** *auto*

fun *initAvstore* :: *avstore* $\Rightarrow$ *bool* **where**
initAvstore (*Avstore as*) = (*as aa1* = (*0*, *nat size-aa1*) $\wedge$ *as aa2* = (*nat size-aa1*, *nat size-aa2*))

fun *istate* ::*state* $\Rightarrow$ *bool* **where**
istate s = (*initAvstore* (*getAvstore s*))

definition *prog* $\equiv$
[
0/ *Start* ,
1/ *Input U xx* ,
2/ *tt* ::= (*N 0*) ,
3/ *IfJump* (*Less* (*V xx*) (*N NN*)) *4 6* ,
4/ *Fence* ,
5/ *tt* ::= (*VA aa2* (*Times* (*VA aa1* (*V xx*)) (*N 512*))),
6/ *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 *xx-NN-cases*: *vs xx* < *int*(*NN*) $\vee$ *vs xx* $\geq$ *int*(*NN*) **by** *auto*

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

using *cases-6*[*of pc*]
by (*auto simp*: *prog-def*)

lemma *eq-Fence-pc*[*simp*]:
pc < *6* ⟹ *prog* ! *pc* = *Fence* ⟷ *pc* = *4*
using *cases-6*[*of pc*]
by (*auto simp*: *prog-def*)

consts *mispred* :: *predState* ⇒ *pcounter list* ⇒ *bool*
fun *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool* **where**
resolve p pc = (*if* (*set pc* = {*6*,*4*}) *then True else False*)

consts *update* :: *predState* ⇒ *pcounter list* ⇒ *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** ‹→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* = *7*
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getUntrustedInput-pcOf*[*simp*]: *pcOf cfg* < *6* ⟹ *is-getInput* (*prog*!(*pcOf cfg*)) ⟷ *pcOf cfg* = *1*
using *cases-6*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *start*[*simp*]:*prog* ! *0* = *Start*
by (*auto simp*: *prog-def*)

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

lemma *if-stat*[*simp*]: *prog*! *3* = (*IfJump* (*Less* (*V xx*) (*N NN*)) *4 6*)
by (*auto simp*: *prog-def*)

lemma *isOutput1*[*simp*]:*prog* ! *6* = *Output U* (*V tt*)
by (*auto simp*: *prog-def*)

lemma *is-Output-pcOf*[*simp*]: *pcOf cfg* < *6* ⟹ *is-Output* (*prog*!(*pcOf cfg*)) ⟷ *pcOf cfg* = *6*
using *cases-6*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *is-Fence-pcOf*[*simp*]: *pcOf cfg* < *6* ⟹(*prog*!(*pcOf cfg*)) = *Fence* ⟷ *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*) ⟷ *pcOf cfg* = *0*
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf*[*simp*]:
isSecO (*pstate*,*cfg*,*cfgs*,*ibT*, *ibUT*,*ls*) ⟷ (*pcOf cfg* = *0* ∧ *cfgs* = [])
using *isSecO-def* **by** *simp*

lemma *getInputT-not*[*simp*]: *pcOf cfg* < *7* ⟹
(*prog* ! *pcOf cfg*) ≠ *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* ⟹
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1 then lhd ibUT else* ⊥)
apply(*subst getActV-simps*) **unfolding** *prog-def*
using *cases-6*[*of pcOf cfg*] **by** *auto*

lemma *getObsV-pcOf*[*simp*]:
pcOf cfg < *7* ⟹
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *6 then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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* ⟹
getActO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1* ∧ *cfgs* = [] *then lhd ibUT else* ⊥)
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* ⟹

getObsO (*pstate*,*cfg*,*cfgs*,*ibT*, *ibUT*,*ls*) =
(*if* (*pcOf cfg* = *6* ∧ *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*
unfolding *prog-def* **by** *auto*

lemma *eqSec-pcOf*[*simp*]:
eqSec (*cfg1*, *ibT*, *ibUT1*, *ls1*) (*pstate3*, *cfg3*, *cfgs3*, *ibT*, *ibUT3*, *ls3*) ⟷
(*pcOf cfg1* = *0* ⟷ *pcOf cfg3* = *0* ∧ *cfgs3* = []) ∧
(*pcOf cfg1* = *0* ⟶ *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 ≠ *LNil* ⟹ *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 ≠ *LNil* ⟹ *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* ⟹
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 ≥ *NN* ⟹
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 ≥ *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 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*:
$pc < 7 \Longrightarrow (pc = 1 \longrightarrow ibUT \neq LNil) \Longrightarrow$
(*Config pc s*, *ibT*, *ibUT*) →*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 < *7* ⟹ (*pc* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹
¬ *finalB* (*Config pc s*, *ibT*, *ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff′*:
pc < *7* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *1* ∧ *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 ≤ *7* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *1* ∧ *ibUT* = *LNil* ∨ *pc* = *7*)
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 ≤ *7* ⟹
finalB (*cfg*, *ibT*, *ibUT*) ⟷ (*pcOf cfg* = *1* ∧ *ibUT* = *LNil* ∨ *pcOf cfg* = *7*)
by (*metis config.collapse finalB-pc-iff*)

lemma *finalS-cond*:*pcOf cfg* < *7* ⟹ *cfgs* = [] ⟹ (*pcOf cfg* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹ ¬ *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-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 cfgs 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 cfgs 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 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 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 cfgs 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 cfgs 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 using *spec-resolve*[*of cfgs pstate cfg update pstate* (*pcOf cfg* # *map pcOf cfgs*) *cfg* [] *ibT ibT ibUT ibUT ls ls*]
by (*metis* (*no-types*, *lifting*) *butlast.simps*(*2*) *empty-set last-ConsL*
length-0-conv length-Suc-conv list.simps(*8*,*9*,*15*) *pos2 resolve.simps*)

subgoal apply(*elim conjE*)
using *spec-resolve*[*of cfgs pstate cfg update pstate* (*pcOf cfg* # *map pcOf cfgs*) *cfg* [] *ibT ibT ibUT ibUT ls ls*]
by (*metis* (*no-types*, *lifting*) *empty-set insert-commute last-ConsL resolve.simps*
length-0-conv length-1-butlast length-Suc-conv list.simps(*9*,*8*,*15*)) **. . . .**

end

9.2 Proof

theory *Fun2-secure*
imports *Fun2*
begin

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

definition *same-xx cfg3 cfgs3 cfg4 cfgs4* ≡

vstore (*getVstore* (*stateOf cfg3*)) *xx* = *vstore* (*getVstore* (*stateOf cfg4*)) *xx* ∧
(∀ *cfg3*′∈*set cfgs3*. *vstore* (*getVstore* (*stateOf cfg3*′)) *xx* = *vstore* (*getVstore* (*stateOf cfg3*)) *xx*) ∧
(∀ *cfg4*′∈*set cfgs4*. *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* ⇒ *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*))) ∧
(∀ *cfg4*′∈*set cfgs4*. *array-base aa2* (*getAvstore* (*stateOf cfg4*′)) = *array-base aa2* (*getAvstore* (*stateOf cfg4*))) ∧
///
(*statA* = *Diff* ⟶ *statO* = *Diff*)))

lemma *common-implies*: *common* (*pstate3*,*cfg3*,*cfgs3*,*ibT*, *ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT*, *ibUT4*,*ls4*)
statA
(*cfg1*,*ibT*, *ibUT1*,*ls1*)

(*cfg2*,*ibT*, *ibUT2*,*ls2*)
statO ⟹
pcOf cfg1 < *8* ∧ *pcOf cfg2* = *pcOf cfg1*
unfolding *common-def PC-def*
by (*auto simp*: *image-def subset-eq*)

definition Δ*0* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*0* = (λ *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 ∧
ibUT1 = *ibUT3* ∧ *ibUT2* = *ibUT4* ∧
(*pcOf cfg3* > *1* ⟶ *same-xx cfg3 cfgs3 cfg4 cfgs4*) ∧
(*pcOf cfg3* < *2* ⟶ *ibUT1*≠*LNil* ∧ *ibUT2*≠*LNil* ∧ *ibUT3*≠*LNil* ∧ *ibUT4*≠*LNil*) ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
pcOf cfg3 ∈ *beforeInput* ∧
noMisSpec cfgs3
))

lemmas Δ*0-defs* = Δ*0-def common-def PC-def*
beforeInput-def
same-xx-def noMisSpec-def

lemma Δ*0-implies*: Δ*0 num*
(*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
(*pcOf cfg3* = *1* ⟶ *ibUT3* ≠ *LNil*) ∧
(*pcOf cfg4* = *1* ⟶ *ibUT4* ≠ *LNil*) ∧
pcOf cfg1 < *7* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
cfgs3 = [] ∧ *pcOf cfg3* < *7* ∧
cfgs4 = [] ∧ *pcOf cfg4* < *7*
unfolding Δ*0-defs*
apply(*intro conjI*)

apply *simp-all*
by (*metis map-is-Nil-conv*)

definition $\Delta 1$ :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 1$ = (λ *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 ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
same-xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 ∈ *afterInput* ∧
noMisSpec cfgs3
))

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

lemma $\Delta 1$*-implies*: $\Delta 1$ *num*
(*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf cfg1 < *7* ∧
cfgs3 = [] ∧ *pcOf cfg3* ≠ *1* ∧ *pcOf cfg3* < *7* ∧
cfgs4 = [] ∧ *pcOf cfg4* ≠ *1* ∧ *pcOf cfg4* < *7*
unfolding $\Delta 1$*-defs*
apply(*intro conjI*) **apply** *simp-all*
apply *linarith*
apply (*metis list.map-disc-iff*)
using *semiring-norm*(*83*,*84*)
by *linarith*

definition $\Delta 2$ :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 2$ = (λ*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 ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
same-xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfThenBranch* ∧
pcOf (*last cfgs3*) = *elseBranch* ∧
misSpecL1 cfgs3
))

lemmas Δ*2-defs* = Δ*2-def common-def PC-def same-xx-def inThenBranch-def elseBranch-def startOfThenBranch-def misSpecL1-def same-xx-def*

lemma Δ*2-implies*: Δ*2 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 cfgs3*) = *6* ∧ *pcOf cfg3* = *4* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *pcOf cfg4* ∧
length cfgs3 = *Suc 0* ∧
length cfgs3 = *length cfgs4*
apply(*intro conjI*)
unfolding Δ*2-defs* **apply** *simp-all*
apply (*simp add*: *image-subset-iff*)
apply (*metis last-map list.map-disc-iff*)
by (*metis length-map*)

definition Δ*3* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*3* = (λ *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 ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
pcOf cfg3 = *elseBranch* ∧
pcOf (*last cfgs3*) = *startOfThenBranch* ∧
same-xx cfg3 cfgs3 cfg4 cfgs4 ∧
misSpecL1 cfgs3
))

lemmas Δ*3-defs* = Δ*3-def common-def PC-def same-xx-def elseBranch-def startOfThenBranch-def*
misSpecL1-def same-xx-def

lemma Δ*3-implies*: Δ*3 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 cfgs3*) = *4* ∧ *pcOf cfg3* = *6* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *pcOf cfg4* ∧
array-base aa1 (*getAvstore* (*stateOf* (*last cfgs3*))) = *array-base aa1* (*getAvstore* (*stateOf cfg3*)) ∧
array-base aa1 (*getAvstore* (*stateOf* (*last cfgs4*))) = *array-base aa1* (*getAvstore* (*stateOf cfg4*)) ∧
length cfgs3 = *Suc 0* ∧
length cfgs3 = *length cfgs4*
apply(*intro conjI*)
unfolding Δ*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*)
by (*metis length-map*)

definition Δ*4* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*4* = (λ*num*
(*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA

(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO.
(*pcOf cfg3* = *endPC* ∧ *pcOf cfg4* = *endPC* ∧ *cfgs3* = [] ∧ *cfgs4* = [] ∧
pcOf cfg1 = *endPC* ∧ *pcOf cfg2* = *endPC*))

lemmas Δ*4-defs* = Δ*4-def common-def endPC-def*

lemma *init*: *initCond* Δ*0*
unfolding *initCond-def*
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 *clarsimp*
apply(*cases getAvstore* (*stateOf cfg3*), *cases getAvstore* (*stateOf cfg4*))
unfolding Δ*0-defs*
unfolding *array-base-def* **by** *auto* **. .**

lemma *step0*: *unwindIntoCond* Δ*0* (*oor* Δ*0* Δ*1*)
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*0*: Δ*0 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 ∆*0 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* ∆*0-defs*
by *simp*

have *f2*:¬*finalN ss2*
using ∆*0 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* ∆*0-defs*
by *simp*

have *f3*:¬*finalS ss3*
using ∆*0* **unfolding** *ss* **apply**−**apply**(*frule* ∆*0-implies*)
using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
using ∆*0* **unfolding** *ss* **apply**−**apply**(*frule* ∆*0-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* ∆*0* ∆*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* ∆*0* ∆*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-defs*)
show *match2* (*oor* ∆*0* ∆*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-defs*)
show *match12* (*oor* ∆*0* ∆*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' 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 = 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* Δ*0* **unfolding** *ss*
by (*simp add*: Δ*0-defs eqSec-def*)

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

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

show *match12-12* (*oor* Δ*0* Δ*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* Δ*0 sstat* **unfolding** *ss cfg statA′* **apply** *simp*
   apply(*simp add*: Δ*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 (*fastforce*)
   using *newStat.simps status.exhaust status.distinct* **by** (*smt*(*z3*))
  } **note** *stat = this*

  show *oor* Δ*0* Δ*1* ∞ *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* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **apply** (*simp add*: Δ*0-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*] **proof**(*cases rule*: *stepS-cases*)
case *nonspec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-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
      qed
    qed
  qed
 qed
qed

lemma step1: unwindIntoCond Δ1 (oor4 Δ1 Δ2 Δ3 Δ4)
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 Δ*1 finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB* Δ*1-defs*
by *simp linarith*

have *f2*:¬*finalN ss2*
using Δ*1 finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB* Δ*1-defs*
by *simp linarith*

have *f3*:¬*finalS ss3*
using Δ*1* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1-implies*)
using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*1* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1-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* Δ*1* Δ*2* Δ*3* Δ*4*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor4* Δ*1* Δ*2* Δ*3* Δ*4*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor4* Δ*1* Δ*2* Δ*3* Δ*4*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor4* Δ*1* Δ*2* Δ*3* Δ*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' 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* Δ*1* **unfolding** *ss*
by (*simp add*: Δ*1-defs eqSec-def*)

show *eqSec ss2 ss4*
using *v sa* Δ*1* **unfolding** *ss*
apply (*simp add*: Δ*1-defs eqSec-def*)
by (*metis length-0-conv length-map*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1* **unfolding** *ss Van.eqAct-def*
apply (*simp-all add*: Δ*1-defs*)
by *linarith*

show *match12-12* (*oor4* Δ*1* Δ*2* Δ*3* Δ*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* Δ*1 sstat* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*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*

show (*oor4* Δ*1* Δ*2* Δ*3* Δ*4*) ∞ *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* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *nonspec-mispred* **note** *nm4* = *nonspec-mispred*
then show *?thesis*
using *sa* Δ*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: Δ1-defs Δ2-defs)
                subgoal apply(rule oor4I3) by (simp add: Δ1-defs Δ3-defs) .
              by (simp add: Δ1-defs)+
          qed
        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: 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 oor4I4) by (simp add: Δ1-defs Δ4-defs)
              subgoal apply(rule oor4I4) by (simp add: Δ1-defs Δ4-defs) .
          qed
        qed
      qed
    qed
  qed
qed
```

lemma *step2*: *unwindIntoCond* Δ*2* Δ*1*
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*2*: Δ*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*:¬*finalN ss1*
using Δ*2 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*2-defs*
by *auto*

have *f2*:¬*finalN ss2*
using Δ*2 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*2-defs*
by *auto*

have *f3*:¬*finalS ss3*
using Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-implies*)
using *finalS-cond-spec* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-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* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*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 ¬ *isSecO ss3*
using *v sa* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)

show ¬ *isSecO ss4*
using *v sa* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)

show *stat*: *statA* = *statA′* ∨ *statO* = *Diff*
using *v sa* Δ*2*
apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
using *v sa* Δ*2* **unfolding** *ss statA′* **apply** *clarsimp*
apply(*simp-all add*: Δ*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*))

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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 v3* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by**(*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
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* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis* **using** *sa stat* Δ*2 v3 v4 sr3 sr4*
unfolding *ss lcfgs hh* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs* Δ*1-defs*, *metis*)
qed
qed

qed
qed
qed

lemma *step3*: *unwindIntoCond* Δ*3* (*oor* Δ*3* Δ*1*)
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*3*: Δ*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*:¬*finalN ss1*
using Δ*3 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*3-defs*
by *auto*

have *f2*:¬*finalN ss2*
using Δ*3 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*3-defs*
by *auto*

have *f3*:¬*finalS ss3*
using Δ*3* **unfolding** *ss* **apply**−**apply**(*frule* Δ*3-implies*)
using *finalS-cond-spec* **by** *simp*

have *f4*:¬*finalS ss4*
  using Δ*3* **unfolding** *ss* **apply**−**apply**(*frule* Δ*3-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* (*oor* Δ*3* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

  show *match1* (*oor* Δ*3* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
  unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
  show *match2* (*oor* Δ*3* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
  unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
  show *match12* (*oor* Δ*3* Δ*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′*

    show ¬ *isSecO ss3*
    using *v sa* Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)

    show ¬ *isSecO ss4*
    using *v sa* Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)

    show *stat*: *statA* = *statA′* ∨ *statO* = *Diff*
    using *v sa* Δ*3*
    apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
    apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
    using *v sa* Δ*3* **unfolding** *ss statA′* **apply** *clarsimp*
    apply(*simp-all add*: Δ*3-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)

show *oor* Δ*3* Δ*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 stat* Δ*3 lcfgs* **unfolding** *ss* **by** (*simp-all add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss* **by** (*simp-all add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss* **apply**−
apply(*frule* Δ*3-implies*) **by** (*simp-all add*: Δ*3-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss* **apply**−
apply(*frule* Δ*3-implies*) **by** (*simp-all add*: Δ*3-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss*
apply (*simp add*: Δ*3-defs*)
by (*metis cfgs-map config.sel*(*1*) *empty-set list.set-map list.simps*(*15*))
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* Δ*3 lcfgs sr3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sr3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sr3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sr3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sr3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)

next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis*
apply(*intro oorI2*)
using *sa stat* Δ*3 lcfgs v3 v4 sr3 sr4* **unfolding** *ss hh*
apply(*simp add*: Δ*3-defs* Δ*1-defs*)
by (*metis empty-iff empty-set length-1-butlast map-eq-imp-length-eq*)
qed
qed
qed
qed
qed

lemma *stepe*: *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* ∧ *finalN ss1* = *finalS ss3* ∧ *finalN ss2* = *finalS ss4*
using Δ*4 Opt.final-def Prog.endPC-def finalS-def stepS-endPC*
unfolding Δ*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 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
   apply(rule match12-simpleI) using Δ4 unfolding ss apply (simp add: Δ4-defs)
    by (simp add: stepS-endPC)
  qed
qed



lemmas theConds = step0 step1 step2 step3 stepe

proposition rsecure
proof−
  define m where m: m ≡ (5::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 Δ3
  else Δ4
  define nxt where nxt: nxt ≡ λi::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 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
```

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* ≡
[
0 Start ,
1 Input U xx ,
2 tt ::= (N 0) ,
3 IfJump (Less (V xx) (N NN)) 4 7 ,
4 vv ::= VA aa1 (V xx) ,
5 Fence ,

6/ tt ::= (VA aa2 (Times (V vv) (N 512))) ,
7/ 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*]:
$pcOf\ cfg < 8 \Longrightarrow is\text{-}IfJump\ (prog\ !\ (pcOf\ cfg)) \longleftrightarrow pcOf\ cfg = 3$
apply(*cases cfg*) **subgoal for** *pc s* **using** *cases-7*[*of pcOf cfg*]
by (*auto simp*: *prog-def*) **.**

lemma *is-If-pc*[*simp*]:
$pc < 8 \Longrightarrow is\text{-}IfJump\ (prog\ !\ pc) \longleftrightarrow pc = 3$
using *cases-7*[*of pc*]
by (*auto simp*: *prog-def*)

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

fun *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool* **where**
resolve p pc = (*if* (*pc* = [4,7]) *then True else False*)

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** ‹→M› *55*)
where *x* →*M y* == *stepM x y*

abbreviation
stepN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *loc set* ⇒ *bool* (**infix** ‹→N› *55*)
where *x* →*N y* == *stepN x y*

abbreviation
stepsN-abbrev :: *config* × *val llist* × *val llist* × *loc set* ⇒ *config* × *val llist* × *val llist* × *loc set* ⇒ *bool* (**infix** ‹→N∗› *55*)
where *x* →*N*∗ *y* == *star stepN x y*

abbreviation
stepS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** ‹→S› *55*)
where *x* →*S y* == *stepS x y*

abbreviation
stepsS-abbrev :: *configS* ⇒ *configS* ⇒ *bool* (**infix** ‹→S∗› *55*)
where *x* →*S*∗ *y* == *star stepS x y*

lemma *endPC*[*simp*]: *endPC* = *8*
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getTrustedInput-pcOf*[*simp*]: *pcOf cfg* < *8* ⟹ *is-getInput* (*prog*!(*pcOf cfg*)) ⟷ *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*]: ¬*is-getInput* (*prog* ! *3*)
by (*auto simp*: *prog-def*)

lemma *getInput-not4*[*simp*]: ¬*is-getInput* (*prog* ! *4*)
by (*auto simp*: *prog-def*)
lemma *Output-not4*[*simp*]: ¬*is-Output* (*prog* ! *4*)
by (*auto simp*: *prog-def*)

lemma *is-Output-pcOf*[*simp*]: *pcOf cfg* < *8* ⟹ *is-Output* (*prog*!(*pcOf cfg*)) ⟷ *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*) (*Avstore as*) (*Heap h*) *p*) ⟹ ¬ *is-getInput* (*prog* ! *pcOf cfg*)
unfolding *is-getInput-def prog-def* **by** *simp*

lemma *not-is-Output*[*simp*]:*cfg* = *Config pc* (*State* (*Vstore vs*) (*Avstore as*) (*Heap h*) *p*) ⟹
pc = *3* ⟹ ¬ *is-Output* (*prog* ! *pcOf cfg*)
unfolding *is-Output prog-def* **by** *simp*

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

lemma *isSecO-pcOf*[*simp*]:
isSecO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟷ (*pcOf cfg* = *0* ∧ *cfgs* = [])
using *isSecO-def* **by** *simp*

lemma *getInputT-not*[*simp*]: *pcOf cfg* < *8* ⟹
(*prog* ! *pcOf cfg*) ≠ *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* ⟹
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =

(*if pcOf cfg* = *1 then lhd ibUT else* ⊥)
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* ⟹
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *7 then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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* ⟹
getActO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1* ∧ *cfgs* = [] *then lhd ibUT else* ⊥)
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* ⟹
getObsO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if* (*pcOf cfg* = *7* ∧ *cfgs* = []) *then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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*) ⟷
(*pcOf cfg1* = *0* ⟷ *pcOf cfg3* = *0* ∧ *cfgs3* = []) ∧
(*pcOf cfg1* = *0* ⟶ *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 ≠ *LNil* ⟹ *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 ≠ *LNil* ⟹ *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* ⟹
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 ≥ *int NN* ⟹
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 ≥ *int 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 < *int NN* ⟹
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* ⟹ (*pc* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹
(*Config pc s*, *ibT*,*ibUT*) →*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* ⟹ (*pc* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹
¬ *finalB* (*Config pc s*, *ibT*,*ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff′*:
pc < *8* ⟹
finalB (*Config pc s*, *ibT*,*ibUT*) ⟷
(*pc* = *1* ∧ *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 ≤ *8* ⟹
finalB (*Config pc s*, *ibT*,*ibUT*) ⟷
(*pc* = *1* ∧ *ibUT* = *LNil* ∨ *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 ≤ *8* ⟹
finalB (*cfg*, *ibT*,*ibUT*) ⟷ (*pcOf cfg* = *1* ∧ *ibUT* = *LNil* ∨ *pcOf cfg* = *8*)
by (*metis config.collapse finalB-pc-iff*)

lemma *finalS-cond*:*pcOf cfg* < *8* ⟹ *cfgs* = [] ⟹ (*pcOf cfg* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹ ¬ *finalS* (*pstate*, *cfg*, *cfgs*, *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 cfgs 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 cfgs 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 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 cfgs 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* ∪ *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* ∪ {*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* ∪ {*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* ⟹
(((*pcOf* (*last cfgs*) = *4* ∨ *pcOf* (*last cfgs*) = *5*) ∧ *pcOf cfg* = *7*) ∨
(*pcOf* (*last cfgs*) = *7* ∧ *pcOf cfg* = *4*)) ⟹
length cfgs = *Suc 0* ⟹
¬ *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*)
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* ∪ *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*)])
apply(*erule notE*) **by**(*rule spec-Fence*, *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

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*))) ∧
 (∀ *cfg4'*∈*set cfgs4*. *array-base aa2* (*getAvstore* (*stateOf cfg4'*)) = *array-base aa2* (*getAvstore* (*stateOf cfg4*))) ∧
 /////
 (*statA* = *Diff* ⟶ *statO* = *Diff*)))

lemma *common-implies*: *common*
  (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
  (*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
  statA
  (*cfg1*,*ibT1*,*ibUT1*,*ls1*)
  (*cfg2*,*ibT2*,*ibUT2*,*ls2*)
  statO ⟹

pcOf cfg1 < *9* ∧ *pcOf cfg2* = *pcOf cfg1*
unfolding *common-def PC-def* **by** (*auto simp*: *image-def subset-eq*)

definition Δ*0* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*0* = (λ*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 ∧
ibUT1 = *ibUT3* ∧ *ibUT2* = *ibUT4* ∧
(*pcOf cfg3* > *1* ⟶ *same-var-o xx cfg3 cfgs3 cfg4 cfgs4*) ∧
(*pcOf cfg3* < *2* ⟶ *ibUT1*≠*LNil* ∧ *ibUT2*≠*LNil* ∧ *ibUT3*≠*LNil* ∧ *ibUT4*≠*LNil*)
∧
pcOf cfg3 ∈ *beforeInput* ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
noMisSpec cfgs3
))

lemmas Δ*0-defs* = Δ*0-def common-def PC-def beforeInput-def noMisSpec-def same-var-o-def*

lemma Δ*0-implies*: Δ*0 num*
(*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
(*pcOf cfg3* = *1* ⟶ *ibUT3* ≠ *LNil*) ∧
(*pcOf cfg4* = *1* ⟶ *ibUT4* ≠ *LNil*) ∧
pcOf cfg1 < *8* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
cfgs3 = [] ∧ *pcOf cfg3* < *8* ∧
cfgs4 = [] ∧ *pcOf cfg4* < *8*
unfolding Δ*0-defs*
apply(*intro conjI*)
apply *simp-all*
by (*metis map-is-Nil-conv*)

definition $\Delta 1$ :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 1$ = (λ*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 ∧
 pcOf cfg3 ∈ *afterInput* ∧
 same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
 ls1 = *ls3* ∧ *ls2* = *ls4* ∧
 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 ⟹
  pcOf cfg1 < *8* ∧
  cfgs3 = [] ∧ *pcOf cfg3* ≠ *1* ∧ *pcOf cfg3* < *8* ∧
  cfgs4 = [] ∧ *pcOf cfg4* ≠ *1* ∧ *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 $\Delta 2$ :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 2$ = (λ*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 ∧
pcOf cfg3 = *startOfThenBranch* ∧
pcOf (*last cfgs3*) = *elseBranch* ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
misSpecL1 cfgs3
))

lemmas Δ*2-defs* = Δ*2-def common-def PC-def same-var-def startOfThenBranch-def*

misSpecL1-def elseBranch-def

lemma Δ*2-implies*: Δ*2 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 cfgs3*) = *7* ∧ *pcOf cfg3* = *4* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *pcOf cfg4* ∧
length cfgs3 = *Suc 0* ∧
length cfgs3 = *length cfgs4*
apply(*intro conjI*)
unfolding Δ*2-defs* **apply** *simp-all*
apply (*simp add*: *image-subset-iff*)
apply (*metis last-map map-is-Nil-conv*)
by (*metis length-map*)

definition Δ*3* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*3* = (λ*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 ∧
pcOf cfg3 = *elseBranch* ∧
pcOf (*last cfgs3*) ∈ *inThenBranchBeforeFence* ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
Language-Prelims.dist ls3 ls4 ⊆ *Language-Prelims.dist ls1 ls2* ∧
(*pcOf* (*last cfgs3*) = *4* ⟶ *ls1* = *ls3* ∧ *ls2* = *ls4*) ∧
misSpecL1 cfgs3
))

lemmas Δ*3-defs* = Δ*3-def common-def PC-def inThenBranchBeforeFence-def beforeAssign-vv-def misSpecL1-def elseBranch-def same-var-o-def*

lemma Δ*3-implies*: Δ*3 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 cfgs3*) = *4* ∨ *pcOf* (*last cfgs3*) = *5*) ∧ *pcOf cfg3* = *7* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *pcOf cfg4* ∧
array-base aa1 (*getAvstore* (*stateOf* (*last cfgs3*))) = *array-base aa1* (*getAvstore* (*stateOf cfg3*)) ∧
array-base aa1 (*getAvstore* (*stateOf* (*last cfgs4*))) = *array-base aa1* (*getAvstore* (*stateOf cfg4*)) ∧
length cfgs3 = *Suc 0* ∧
length cfgs3 = *length cfgs4* ∧
vstore (*getVstore* (*stateOf* (*last cfgs3*))) *xx* = *vstore* (*getVstore* (*stateOf* (*last cfgs4*))) *xx*
apply(*intro conjI*)
unfolding Δ*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 Δ*1'* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status*

⇒ *bool* **where**
Δ*1′* = (λ*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 ∧
 pcOf cfg3 = *elseBranch* ∧
 same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
 Language-Prelims.dist ls3 ls4 ⊆ *Language-Prelims.dist ls1 ls2* ∧
 noMisSpec cfgs3
))

lemmas Δ*1′-defs* = Δ*1′-def common-def PC-def afterInput-def same-var-o-def*
noMisSpec-def
elseBranch-def

lemma Δ*1′-implies*: Δ*1′ num*
 (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
 (*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
 statA
 (*cfg1*,*ibT1*,*ibUT1*,*ls1*)
 (*cfg2*,*ibT2*,*ibUT2*,*ls2*)
 statO ⟹
 pcOf cfg1 < *8* ∧
 cfgs3 = [] ∧ *pcOf cfg3* ≠ *1* ∧ *pcOf cfg3* < *8* ∧
 cfgs4 = [] ∧ *pcOf cfg4* ≠ *1* ∧ *pcOf cfg4* < *8*
 unfolding Δ*1′-defs*
 apply(*intro conjI*) **apply** *simp-all*
 by (*metis list.map-disc-iff*)

definition Δ*4* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status*
⇒ *bool* **where**
Δ*4* = (λ*num*
 (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
 (*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
 statA
 (*cfg1*,*ibT1*,*ibUT1*,*ls1*)
 (*cfg2*,*ibT2*,*ibUT2*,*ls2*)

statO.
(*pcOf cfg3* = *endPC* ∧ *pcOf cfg4* = *endPC* ∧ *cfgs3* = [] ∧ *cfgs4* = [] ∧
pcOf cfg1 = *endPC* ∧ *pcOf cfg2* = *endPC*))

lemmas Δ*4-defs* = Δ*4-def common-def endPC-def*

lemma *init*: *initCond* Δ*0*
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 *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 Δ*0-defs array-base-def* **by** *auto* **. .**

lemma *step0*: *unwindIntoCond* Δ*0* (*oor* Δ*0* Δ*1*)
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*0*: Δ*0 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 Δ*0 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*0-defs*
by *simp*

have *f2*:¬*finalN ss2*
using Δ*0 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*0-defs*
by *simp*

have *f3*:¬*finalS ss3*
using Δ*0* **unfolding** *ss* **apply**−**apply**(*frule* Δ*0-implies*)
using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*0* **unfolding** *ss* **apply**−**apply**(*frule* Δ*0-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* Δ*0* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* Δ*0* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-defs*)
show *match2* (*oor* Δ*0* Δ*1*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-defs*)
show *match12* (*oor* Δ*0* Δ*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' 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* = *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* Δ*0* **unfolding** *ss*
by (*simp add*: Δ*0-defs eqSec-def*)

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

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

show *match12-12* (*oor* Δ*0* Δ*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* Δ*0 sstat* **unfolding** *ss cfg statA′* **apply** *simp*
apply(*simp add*: Δ*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*

show *oor* Δ*0* Δ*1* ∞ *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* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **apply** (*simp add*: Δ*0-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*] **proof**(*cases rule*: *stepS-cases*)
case *nonspec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *nonspec-normal* **note** *nn4*=*nonspec-normal*
show *?thesis* **using** *sa stat* Δ*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: 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
      qed
    qed
  qed
 qed
qed

lemma step1: unwindIntoCond Δ1 (oor4 Δ1 Δ2 Δ3 Δ4)
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 Δ1 finalB-pc-iff' unfolding ss cfg finalN-iff-finalB Δ1-defs
    by simp linarith

  have f2:¬finalN ss2
    using Δ1 finalB-pc-iff' unfolding ss cfg finalN-iff-finalB Δ1-defs
    by simp linarith


  have f3:¬finalS ss3
    using Δ1 unfolding ss apply−apply(frule Δ1-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' 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* Δ*1* **unfolding** *ss*
by (*simp add*: Δ*1-defs eqSec-def*)

show *eqSec ss2 ss4*
using *v sa* Δ*1* **unfolding** *ss*
apply (*simp add*: Δ*1-defs eqSec-def*)
by (*metis length-0-conv length-map*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1* **unfolding** *ss Van.eqAct-def*
apply (*simp-all add*: Δ*1-defs*)
by *linarith*

show *match12-12* (*oor4* Δ*1* Δ*2* Δ*3* Δ*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* Δ*1 sstat* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*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*

show (*oor4* Δ*1* Δ*2* Δ*3* Δ*4*) ∞ *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* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *nonspec-mispred* **note** *nm4* = *nonspec-mispred*
then show *?thesis*
using *sa* Δ*1 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*: Δ*1-defs* Δ*2-defs*)
subgoal apply(*rule oor4I3*) **by** (*simp add*: Δ*1-defs* Δ*3-defs*) **.**
by (*simp-all add*: Δ*1-defs*)+
qed
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* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *nonspec-normal*
then show *?thesis* **using** *sa* Δ*1 stat v3 v4 nn3* **unfolding** *ss cfg hh* **apply** *clarsimp*
using *cases-7*[*of pc3*] **apply**(*elim disjE*)
subgoal by (*simp add*: Δ*1-defs*)
subgoal by (*simp add*: Δ*1-defs*)
subgoal apply(*rule oor4I1*) **by**(*simp add*:Δ*1-defs*)
subgoal using *xx-NN-cases*[*of vs3*] **apply**(*elim disjE*)
subgoal apply(*rule oor4I1*) **by** (*simp add*: Δ*1-defs*)
subgoal apply(*rule oor4I1*) **by** (*simp add*: Δ*1-defs*) **.**
subgoal apply(*rule oor4I1*) **by** (*simp add*: Δ*1-defs*)
subgoal apply(*rule oor4I1*) **by** (*simp add*: Δ*1-defs*)
subgoal apply(*rule oor4I1*) **by** (*simp add*: Δ*1-defs*)
subgoal apply(*rule oor4I4*) **by** (*simp add*: Δ*1-defs* Δ*4-defs*)
subgoal apply(*rule oor4I4*) **by** (*simp add*: Δ*1-defs* Δ*4-defs*) **.**
qed
qed
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* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*1 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*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 ¬ *isSecO ss3*
using *v sa* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)

show ¬ *isSecO ss4*
using *v sa* Δ*2* **unfolding** *ss* **apply** *clarsimp*
by (*simp add*: Δ*2-defs*, *linarith*)

show *stat*: *statA* = *statA′* ∨ *statO* = *Diff*
using *v sa* Δ*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*: Δ*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*))

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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 v3* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by**(*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
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* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis* **using** *sa stat* Δ*2 v3 v4 sr3 sr4*
unfolding *ss lcfgs hh* **apply**−
apply(*frule* Δ*2-implies*) **apply** (*simp add*: Δ*2-defs* Δ*1-defs*) **by** *clarsimp*
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 Δ3: Δ3 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:¬finalN ss1
    using Δ3 finalB-pc-iff' unfolding ss finalN-iff-finalB Δ3-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 Δ*3* **unfolding** *ss* **apply**−**apply**(*frule* Δ*3-implies*)
using *finalS-cond-spec* **by** *simp*

have *vs3 xx* = *vs4 xx*
using Δ*3 lcfgs* **unfolding** *ss*
apply−**by**(*frule* Δ*3-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* Δ*3* Δ*1*′) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* Δ*3* Δ*1*′) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor* Δ*3* Δ*1*′) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor* Δ*3* Δ*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*′

show ¬ *isSecO ss3*
using *v sa* Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)

show ¬ *isSecO ss4*
using *v sa* Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)

show *stat*: *statA* = *statA*′ ∨ *statO* = *Diff*
using *v sa* Δ*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*: Δ*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)

show *oor* Δ*3* Δ*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 stat* Δ*3 lcfgs* **unfolding** *ss* **by** (*simp-all add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss* **by** (*simp-all add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs* **unfolding** *ss* **apply**−
apply(*frule* Δ*3-implies*, *clarsimp*)
by (*auto simp add*: Δ*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* Δ*3 lcfgs sn3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sn3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sn3* **unfolding** *ss*
apply (*simp add*: Δ*3-defs*)
by (*metis config.sel*(*1*) *last-map*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sn3* **unfolding** *ss*
apply (*simp add*: Δ*3-defs*)
by (*metis config.sel*(*1*) *last-map*)
next
case *spec-resolve*
then show *?thesis* **using** *sa stat* Δ*3 lcfgs sn3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*)
next
case *spec-normal* **note** *sn4* = *spec-normal*
show *?thesis*
apply(*intro oorI1*)

```
unfolding ss Δ3-def apply− apply(clarify,intro conjI)
  subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
  apply− apply(frule Δ3-implies) apply(simp add: Δ3-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 Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
  apply− apply(frule Δ3-implies) by(simp add: Δ3-defs)
  subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
  apply− apply(frule Δ3-implies) apply(simp add: Δ3-defs)
  using cases-7[of pc3] apply simp apply(elim disjE)
  by simp-all
  subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
  apply− apply(frule Δ3-implies) apply(simp add: Δ3-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 Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
    apply− apply(frule Δ3-implies) apply(simp add: Δ3-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 Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
    apply− apply(frule Δ3-implies) apply(simp add: Δ3-defs)
    using cases-7[of pc3] by(elim disjE, simp-all)
  subgoal using sa stat Δ3 lcfgs v3 v4 sn3 sn4 unfolding ss hh
  apply− apply(frule Δ3-implies) apply(simp-all add: Δ3-defs)
    by (metis length-Suc-conv list.size(3)) .
qed
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 Δ3 lcfgs sf3 unfolding ss
    by (simp add: Δ3-defs)
  next
    case nonspec-mispred
    then show ?thesis using sa stat Δ3 lcfgs sf3 unfolding ss
    by (simp add: Δ3-defs)
  next
    case spec-mispred
    then show ?thesis using sa stat Δ3 lcfgs sf3 unfolding ss
    apply (simp add: Δ3-defs)
    by (metis com.disc config.sel(1) last-map)
  next
    case spec-resolve
    then show ?thesis using sa stat Δ3 lcfgs sf3 unfolding ss
    by (simp add: Δ3-defs)
```

```
        next
          case spec-normal
          then show ?thesis using sa stat Δ3 lcfgs sf3 unfolding ss
          apply (simp add: Δ3-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 Δ1'-defs
          using sa stat Δ3 lcfgs v3 v4 sf3 sf4 unfolding ss hh
          apply− by(simp-all add: Δ3-defs Δ1'-defs, blast)
        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-resolve note sr4 = spec-resolve
          show ?thesis
          apply(intro oorI2)
          using sa stat Δ3 lcfgs v3 v4 sr3 sr4 unfolding ss hh
          by(simp add: Δ3-defs Δ1-defs)
        qed
      qed
    qed
  qed
qed
```

lemma *step1′*: *unwindIntoCond* Δ*1′* 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*1′*: Δ*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*:¬*finalN ss1*
using Δ*1′ finalB-pc-iff′* **unfolding** *ss cfg finalN-iff-finalB* Δ*1′-defs*
by *simp*

have *f2*:¬*finalN ss2*
using Δ*1′ finalB-pc-iff′* **unfolding** *ss cfg finalN-iff-finalB* Δ*1′-defs*
by *simp*

have *f3*:¬*finalS ss3*
using Δ*1′* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1′-implies*)
using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*1′* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1′-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* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*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′ 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* Δ*1′* **unfolding** *ss*
by (*simp add*: Δ*1′-defs eqSec-def*)

show *eqSec ss2 ss4*
using *v sa* Δ*1′* **unfolding** *ss*
by (*simp add*: Δ*1′-defs eqSec-def*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1′* **unfolding** *ss Van.eqAct-def*
by (*simp-all add*: Δ*1′-defs*)

show *match12-12* Δ*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*

show $\Delta 4$ ∞ $ss3'$ $ss4'$ $statA'$ (*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** *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*
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
qed
qed
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* ∧ *finalN ss1* = *finalS ss3* ∧ *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* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*4 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*4 ss3 ss4 statA ss1 ss2 statO*
apply(*rule match12-simpleI*) **using** Δ*4* **unfolding** *ss* **apply** (*simp add*: Δ*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* ≡ (*6*::*nat*)
define Δ*s* **where** Δ*s*: Δ*s* ≡ λ*i*::*nat*.
if i = 0 then Δ*0*
else if i = 1 then Δ*1*
else if i = 2 then Δ*2*
else if i = 3 then Δ*3*
else if i = 4 then Δ*4*
else Δ*1'*
define *nxt* **where** *nxt*: *nxt* ≡ λ*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 nxt* Δ*s*])
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 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* (‹⊥›)

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 ≠ *aa2 aa1* ≠ *vv aa1* ≠ *xx aa1* ≠ *tt*
aa2 ≠ *aa1 aa2* ≠ *vv aa2* ≠ *xx aa1* ≠ *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* ≡
[
0/ *Start* ,
1/ *Input U xx* ,
2/ *tt* ::= (*N 0*) ,
3/ *IfJump* (*Less* (*V xx*) (*N NN*)) *4 6* ,
4/   *vv* ::= *VA aa1* (*N 0*) ,
5/   *tt* ::= *Plus* (*VA aa2* (*Times* (*V vv*) (*N 512*))) (*V xx*) ,
6/ *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 *cases-thenBranch*: (*i*::*pcounter*) < *4* ∨ *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* ∨ *vs xx* ≥ *int NN* **by** *auto*

lemma *is-If-pcOf*[*simp*]:
pcOf cfg < *7* ⟹ *is-IfJump* (*prog* ! (*pcOf cfg*)) ⟷ *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* ⟹ *is-IfJump* (*prog* ! *pc*) ⟷ *pc* = *3*

using *cases-6*[*of pc*]
by (*auto simp*: *prog-def*)

lemma *is-If-pcThen*[*simp*]: *pcOf cfg* ∈ {*4..6*} ⟹ ¬*is-IfJump* (*prog* ! *pcOf cfg*)
using *cases-thenBranch*[*of pcOf cfg*]
by (*auto simp*: *prog-def*)

consts *mispred* :: *predState* ⇒ *pcounter list* ⇒ *bool*
fun *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool* **where**
resolve p pc = (*if* (*pc* = [*6*,*6*] ∨ *pc* = [*4*,*6*]) *then True else False*)

consts *update* :: *predState* ⇒ *pcounter list* ⇒ *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** ‹→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* = *7*
unfolding *endPC-def* **unfolding** *prog-def* **by** *auto*

lemma *is-getUntrustedInput-pcOf*[*simp*]: *pcOf cfg* < *7* ⟹ *is-getInput* (*prog*!(*pcOf cfg*)) ⟷ *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*]:¬*is-getInput* (*prog* ! *4*)
unfolding *prog-def* **by** *auto*
lemma *getInput-not5*[*simp*]:¬*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* ⟹ *is-Output* (*prog*!(*pcOf cfg*)) ⟷ *pcOf cfg* = *6*
using *cases-6*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *is-Fence-pcOf*[*simp*]:*pcOf cfg* < *7* ⟹ *prog* ! (*pcOf cfg*) ≠ *Fence*
using *cases-6*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *is-Fence-pcThen*[*simp*]: *3* ≤ *pcOf cfg* ∧ *pcOf cfg* ≤ *5* ⟹ (*prog* ! *pcOf cfg*)≠*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*) ⟹ *False* **unfolding** *prog-def* **by** *simp*
lemma *Output-not4*[*intro*]:*is-Output* (*prog* ! *4*) ⟹ *False* **unfolding** *prog-def* **by** *simp*
lemma *Fence-not4*[*intro*]:*prog* ! *4* = *Fence* ⟹ *False* **unfolding** *prog-def* **by** *simp*

lemma *getInput-not55*[*intro*]:*is-getInput* (*prog* ! *5*) ⟹ *False* **unfolding** *prog-def* **by** *simp*
lemma *Output-not5*[*intro*]:*is-Output* (*prog* ! *5*) ⟹ *False* **unfolding** *prog-def* **by** *simp*
lemma *Fence-not5*[*intro*]:*prog* ! *5* = *Fence* ⟹ *False* **unfolding** *prog-def* **by** *simp*

lemma *Jump-not6*:¬ *is-IfJump* (*prog* ! *6*)**unfolding** *prog-def* **by** *simp*

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

lemma *isSecO-pcOf*[*simp*]:
isSecO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟷ (*pcOf cfg* = *0* ∧ *cfgs* = [])
using *isSecO-def* **by** *simp*

lemma *inputT-not*[*simp*]: *pcOf cfg* < *7* ⟹
(*prog* ! *pcOf cfg*) ≠ *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* ⟹
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1 then lhd ibUT else* ⊥)
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* ⟹
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *6 then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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* $\Longrightarrow$
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)

by *simp*

lemma *getActO-pcOf*[*simp*]:
pcOf cfg < *7* $\Longrightarrow$
getActO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *1* ∧ *cfgs* = [] *then lhd ibUT else* ⊥)
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* $\Longrightarrow$
getObsO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if* (*pcOf cfg* = *6* ∧ *cfgs* = []) *then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
apply(*subst getObsO-simps*)
apply(*cases cfgs*, *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*, *cfg3*, *cfgs3*, *ibT3*, *ibUT3*, *ls3*) ⟷
(*pcOf cfg1* = *0* ⟷ *pcOf cfg3* = *0* ∧ *cfgs3* = []) ∧
(*pcOf cfg1* = *0* ⟶ *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 ≠ *LNil* ⟹ *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 ≠ *LNil* ⟹ *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* ⟹
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 ≥ *int NN* ⟹
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 ≥ *int 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 < *int NN* ⟹
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* ⟹ (*pc* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹
(*Config pc s*, *ibT*,*ibUT*) →*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*:
$4 \leq pc \wedge pc \leq 6 \Longrightarrow$
$(nextB\ (Config\ pc\ (State\ (Vstore\ vs)\ avst\ (Heap\ h)\ p), ibT, ibUT)) = (Config\ pc'$
$(State\ (Vstore\ vs')\ avst'\ (Heap\ h')\ p'), ibT, ibUT') \Longrightarrow$
$avst = avst' \wedge$
$vs\ xx = vs'\ xx \wedge$
$h = h'$
using *cases-thenBranch*[*of pc*]
apply *safe*
apply *simp-all* **by** *auto*

lemma *nextB-avst-consistent*:
$4 \leq pcOf\ cfg \wedge pcOf\ cfg \leq 6 \Longrightarrow$
$(nextB\ (cfg, ibT, ibUT)) = (cfg', ibT, ibUT') \Longrightarrow$
$(getAvstore\ (stateOf\ cfg)) = (getAvstore\ (stateOf\ cfg')) \wedge$
$(getHheap\ (stateOf\ cfg)) = (getHheap\ (stateOf\ cfg')) \wedge$
$vstore\ (getVstore\ (stateOf\ cfg))\ xx = vstore\ (getVstore\ (stateOf\ cfg'))\ xx$
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 ≤ *pcOf cfg1* ∧ *pcOf cfg1* ≤ *6* ⟹ *pcOf cfg1* = *pcOf cfg2* ⟹
(*nextB* (*cfg1*, *ibT1*, *ibUT1*)) = (*cfg1′*, *ibT1′*, *ibUT1′*) ⟹
(*nextB* (*cfg2*, *ibT2*, *ibUT2*)) = (*cfg2′*, *ibT2′*, *ibUT2′*) ⟹
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 < *7* ⟹ (*pc* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹
¬ *finalB* (*Config pc s*, *ibT*, *ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff′*:
pc < *7* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *1* ∧ *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 ≤ *7* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *1* ∧ *ibUT* = *LNil* ∨ *pc* = *7*)
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 ≤ *7* ⟹
finalB (*cfg*, *ibT*, *ibUT*) ⟷ (*pcOf cfg* = *1* ∧ *ibUT* = *LNil* ∨ *pcOf cfg* = *7*)
by (*metis config.exhaust config.sel*(*1*) *finalB-pc-iff*)

lemma *finalS-cond*:*pcOf cfg* < *7* ⟹ *cfgs* = [] ⟹ (*pcOf cfg* = *1* ⟶ *ibUT* ≠ *LNil*) ⟹ ¬ *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-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 cfgs 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 cfgs 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 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 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 cfgs 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 cfgs Config pc* (*State* (*Vstore vs*) *avst hh p*)
pstate pstate ibT ibUT
(*let l* = (*array-loc aa1 0 avst*)
in (*Config 5* (*State* (*Vstore* (*vs*(*vv* := *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
(*let l* = (*array-loc aa2* (*nat* (*vs vv* ∗ *512*)) *avst*)
in (*Config 6* (*State* (*Vstore* (*vs*(*tt* := *h l* + *vs xx*))) *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* (*last cfgs*) = *5* ∨ *pcOf* (*last cfgs*) = *6*) ∧ *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**(*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(*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* ∪ *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*
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 ∪ 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 (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* ≡ {*0..6*}

definition *same-xx-cp cfg1 cfg2* ≡
vstore (getVstore (stateOf cfg1)) xx = vstore (getVstore (stateOf cfg2)) xx
∧ *vstore (getVstore (stateOf cfg1)) xx = 0*

definition *common-memory cfg cfg′ cfgs′* ≡
array-base aa1 (getAvstore (stateOf cfg)) = array-base aa1 (getAvstore (stateOf cfg′)) ∧
(∀ *cfg′′∈set cfgs′. array-base aa1 (getAvstore (stateOf cfg′′)) = array-base aa1 (getAvstore (stateOf cfg)))* ∧

array-base aa2 (getAvstore (stateOf cfg)) = array-base aa2 (getAvstore (stateOf cfg′)) ∧
(∀ cfg′′∈set cfgs′. array-base aa2 (getAvstore (stateOf cfg′′)) = array-base aa2 (getAvstore (stateOf cfg))) ∧
(getHheap (stateOf cfg)) = (getHheap (stateOf cfg′)) ∧
(∀ cfg′′∈set cfgs′. getHheap (stateOf cfg) = (getHheap (stateOf cfg′′))) ∧
(getAvstore (stateOf cfg)) = (getAvstore (stateOf cfg′))

definition *beforeInput* = {0,1}
definition *afterInput* = {2..6}
definition *elseBranch* = 6
definition *startOfThenBranch* = 4
definition *inThenBranch* = {4..6}

definition *afterInputNotInElse* = {2,3,4,5,6,8}
definition *inThenBranchBeforeOutput* = {3,4,5}
definition *atCond* = 3
definition *atThenOutput* = 5
definition *atJump* = 6

definition *common-strat1* :: stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status ⇒ 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 ∧
cfg1 = cfg3 ∧ cfg2 = cfg4 ∧
pcOf cfg3 = pcOf cfg4 ∧ map pcOf cfgs3 = map pcOf cfgs4 ∧
pcOf cfg3 ∈ PC ∧ pcOf ‘ (set cfgs3) ⊆ PC ∧

~~tr1 and tr3 have common memory~~
common-memory cfg1 cfg3 cfgs3 ∧

~~likewise tr2 and tr4~~
common-memory cfg2 cfg4 cfgs4 ∧

(∀ 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))) ∧
///

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))) ∧
(∀ cfg4′∈set cfgs4. 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*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO.
(*pstate3 = pstate4* ∧

(*num* = (*endPC* − *pcOf cfg1*) ∨ *num* = ∞) ∧

tr1 and tr2 stay on the same path
pcOf cfg1 = pcOf cfg2 ∧

likewise tr3 and tr4
pcOf cfg3 = pcOf cfg4 ∧
map pcOf cfgs3 = map pcOf cfgs4 ∧
pcOf cfg3 ∈ *PC* ∧ *pcOf* ‘ (*set cfgs3*) ⊆ *PC* ∧
pcOf cfg1 ∈ *PC* ∧

tr1 and tr3 have common memory
common-memory cfg1 cfg3 cfgs3 ∧

likewise tr2 and tr4
common-memory cfg2 cfg4 cfgs4 ∧

(∀ *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 ALOT. Maybe this is also worth being extracted as a predicate.~~
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))) ∧
(∀ cfg4'∈set cfgs4. 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,cfgs3,ibT3,ibUT3,ls3)
(pstate4,cfg4,cfgs4,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 Δ0 :: enat ⇒ stateO ⇒ stateO ⇒ status ⇒ stateV ⇒ stateV ⇒ status ⇒ bool **where**
Δ0 = (λnum (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 ∧

~~*head of the buffers are equal counterpartwise*~~
(*llength ibUT1* = ∞ ∧ *llength ibUT2* = ∞ ∧
llength ibUT3 = ∞ ∧ *llength ibUT4* = ∞) ∧
(*lhd ibUT3* ≥ *NN* ∧ (*lhd ibUT1* = *0*) ∧ *ibUT1* = *ibUT2*
∨*lhd ibUT3* < *NN* ∧ *ibUT1* = *ibUT3* ∧ *ibUT2* = *ibUT4*) ∧
pcOf cfg3 ∈ *beforeInput* ∧

~~*since memory is equal, cfg1=cfg3 and cfg2=cfg4 in 'loading' state*~~
cfg1 = *cfg3* ∧ *cfg2* = *cfg4* ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
ls1 ={} ∧ *ls2*={} ∧
noMisSpec cfgs3
))
lemmas Δ*0-defs′* = Δ*0-def common-defs PC-def beforeInput-def noMisSpec-def*

lemma Δ*0-def2*:
Δ*0 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 ∧

~~*head of the buffers are equal counterpartwise*~~
(*llength ibUT1* = ∞ ∧ *llength ibUT2* = ∞ ∧
llength ibUT3 = ∞ ∧ *llength ibUT4* = ∞) ∧
(*ibUT1* ≠ [[]] ∧ *ibUT2* ≠ [[]] ∧ *ibUT3* ≠ [[]] ∧ *ibUT4* ≠ [[]]) ∧
(*lhd ibUT3* ≥ *NN* ∧ (*lhd ibUT1* = *0*) ∧ *ibUT1* = *ibUT2*
∨*lhd ibUT3* < *NN* ∧ *ibUT1* = *ibUT3* ∧ *ibUT2* = *ibUT4*) ∧
pcOf cfg3 ∈ *beforeInput* ∧

~~*since memory is equal, cfg1=cfg3 and cfg2=cfg4 in 'loading' state*~~
cfg1 = *cfg3* ∧ *cfg2* = *cfg4* ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧
ls1 ={} ∧ *ls2*={} ∧
noMisSpec cfgs3

)
unfolding Δ*0-defs'* **apply**(*clarsimp*, *standard*)
subgoal by (*smt* (*verit*) *infinity-ne-i0 llength-LNil*)
subgoal by (*smt* (*verit*)) **.**

lemmas Δ*0-defs* = Δ*0-def2 common-defs PC-def beforeInput-def noMisSpec-def*

lemma Δ*0-implies*: Δ*0 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 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 Δ*0-defs*
apply(*intro conjI*)
apply (*simp-all*)
by (*metis Nil-is-map-conv*)

definition Δ*1* :: *enat* $\Rightarrow$ *stateO* $\Rightarrow$ *stateO* $\Rightarrow$ *status* $\Rightarrow$ *stateV* $\Rightarrow$ *stateV* $\Rightarrow$ *status* $\Rightarrow$ *bool* **where**
Δ*1* = (λ*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 Δ*1-defs* = Δ*1-def common-strat1-defs PC-def afterInput-def same-var-o-def* *noMisSpec-def*

lemma Δ*1-implies*: Δ*1 num*
  (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
  (*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
  statA
  (*cfg1*,*ibT1*,*ibUT1*,*ls1*)
  (*cfg2*,*ibT2*,*ibUT2*,*ls2*)
  statO ⟹
  pcOf cfg1 < *7* ∧
  cfgs3 = [] ∧ *pcOf cfg3* ≠ *1* ∧ *pcOf cfg3* < *7* ∧
  cfgs4 = [] ∧ *pcOf cfg4* ≠ *1* ∧ *pcOf cfg4* < *7*
 unfolding Δ*1-defs*
 apply(*intro conjI*) **apply** *simp-all*
 by (*metis map-is-Nil-conv*)

definition Δ*2* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*2* = (λ*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 ∧
 pcOf cfg3 = *startOfThenBranch* ∧
 pcOf cfg1 = *pcOf cfg3* ∧

 pcOf (*last cfgs3*) = *elseBranch* ∧
 same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
 vstore (*getVstore* (*stateOf cfg3*)) *xx* < *NN* ∧
 ls1 = *ls3* ∧ *ls2* = *ls4* ∧
 misSpecL1 cfgs3
))

lemmas Δ*2-defs* = Δ*2-def common-strat1-defs PC-def same-var-def startOfThenBranch-def*
  misSpecL1-def elseBranch-def

lemma Δ*2-implies*: Δ*2 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 cfgs3*) = *6* ∧ *pcOf cfg3* = *4* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *pcOf cfg4* ∧
length cfgs3 = *Suc 0* ∧
length cfgs3 = *length cfgs4*
apply(*intro conjI*)
unfolding Δ*2-defs* **apply** *simp-all*
apply (*metis last-map map-is-Nil-conv*)
by (*metis length-map*)

definition Δ*1'* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*1'* = (λ*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 ∧
///
pcOf cfg3 ∈ *afterInput* ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧

(*pcOf cfg1* > *2* ⟶ *vstore* (*getVstore* (*stateOf cfg3*)) *tt* = *vstore* (*getVstore* (*stateOf cfg4*)) *tt*) ∧

vstore (*getVstore* (*stateOf cfg3*)) *xx* ≥ *NN* ∧

(*pcOf cfg1* < *4* ⟶ *pcOf cfg1* = *pcOf cfg3* ∧
ls1 = {} ∧ *ls2* = {} ∧
ls1 = *ls3* ∧ *ls2* = *ls4*) ∧
(*pcOf cfg1* ≤ *5* ⟶ *ls1* ⊆ {*array-loc aa1 0* (*getAvstore* (*stateOf cfg1*))}
∧ *ls1* = *ls2* ∧ *ls3* = *ls4*) ∧

(*Language-Prelims.dist ls3 ls4* ⊆ *Language-Prelims.dist ls1 ls2*) ∧

(*pcOf cfg1* ≥ *4* ⟶ *pcOf cfg1* ∈ *inThenBranch* ∧ *pcOf cfg3* = *elseBranch*) ∧
same-xx-cp cfg1 cfg2 ∧
vstore (*getVstore* (*stateOf cfg1*)) *xx* = *0* ∧

ls3 ⊆ *ls1* ∧ *ls4* ⊆ *ls2* ∧
noMisSpec cfgs3
))
lemmas Δ*1′-defs* = Δ*1′-def common-defs PC-def afterInput-def*
same-var-o-def same-xx-cp-def noMisSpec-def inThenBranch-def elseBranch-def
lemma Δ*1′-implies*: Δ*1′ num* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf cfg1 < *7* ∧ *pcOf cfg1* ≠ *Suc 0* ∧
pcOf cfg2 = *pcOf cfg1* ∧
cfgs3 = [] ∧ *pcOf cfg3* < *7* ∧
cfgs4 = [] ∧ *pcOf cfg4* < *7*
unfolding Δ*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 Δ*3′* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*3′* = (λ *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 ∧
///
pcOf cfg3 = *elseBranch* ∧ *cfgs3* ≠ [] ∧
pcOf (*last cfgs3*) ∈ *inThenBranch* ∧
pcOf (*last cfgs4*) = *pcOf*(*last cfgs3*) ∧
~~*/// we sync the cp traces with the speculation cfgs3 || cfgs4*~~
pcOf cfg1 = *pcOf*(*last cfgs3*) ∧

same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
(*getAvstore* (*stateOf cfg3*)) = (*getAvstore* (*stateOf* (*last cfgs3*))) ∧
(*getAvstore* (*stateOf cfg4*)) = (*getAvstore* (*stateOf* (*last cfgs4*))) ∧

same-xx-cp cfg1 cfg2 ∧
ls1 = *ls3* ∧ *ls2* = *ls4* ∧

vstore (*getVstore* (*stateOf cfg3*)) *tt* = *vstore* (*getVstore* (*stateOf cfg4*)) *tt* ∧

vstore (*getVstore* (*stateOf cfg3*)) *xx* ≥ *NN* ∧

(*pcOf cfg1* = *4* ⟶ *ls1* = {} ∧ *ls2* = {}) ∧
(*pcOf cfg1* ≤ *5* ⟶ *ls1* ⊆ {*array-loc aa1 0* (*getAvstore* (*stateOf cfg1*))}
∧ *ls2* ⊆ {*array-loc aa1 0* (*getAvstore* (*stateOf cfg2*))}
∧ *ls3* = *ls4*) ∧

(*pcOf cfg1* > *4* ⟶*same-var vv cfg1* (*last cfgs3*) ∧ *same-var vv cfg2* (*last cfgs4*))
∧
misSpecL1 cfgs3
))

lemmas Δ*3′-defs* = Δ*3′-def common-defs PC-def elseBranch-def*
inThenBranch-def startOfThenBranch-def
same-var-o-def same-xx-cp-def misSpecL1-def same-var-def

lemma Δ*3′-implies*: Δ*3′ num* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf cfg1 < *7* ∧ *pcOf cfg1* ≠ *Suc 0* ∧
pcOf cfg2 = *pcOf cfg1* ∧
pcOf cfg3 < *7* ∧ *pcOf cfg4* < *7* ∧
(*pcOf* (*last cfgs3*) = *4* ∨ *pcOf* (*last cfgs3*) = *5* ∨ *pcOf* (*last cfgs3*) = *6*) ∧ *pcOf cfg3* = *6*
unfolding Δ*3′-defs*
apply(*intro conjI*)
apply *simp-all*
by (*metis cases-thenBranch le-neq-implies-less less-SucI not-less-eq*)

definition Δ*e* :: *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*e* = (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 Δ*0 finalB-pc-iff′* **unfolding** *ss finalN-iff-finalB* Δ*0-defs*
by *simp*

have *f2*:¬*finalN ss2*
using Δ*0 finalB-pc-iff′* **unfolding** *ss finalN-iff-finalB* Δ*0-defs*
by *simp*

have *f3*:¬*finalS ss3*
using Δ*0* **unfolding** *ss* **apply**−**apply**(*frule* Δ*0-implies*)
using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*0* **unfolding** *ss* **apply**−**apply**(*frule* Δ*0-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* Δ*0* Δ*1* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor3* Δ*0* Δ*1* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor3* Δ*0* Δ*1* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor3* Δ*0* Δ*1* Δ*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' 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* = *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* Δ*0* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)

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

show *saO*: *Van.eqAct ss1 ss2*
using *v sa* Δ*0* **unfolding** *ss*
unfolding *Opt.eqAct-def Van.eqAct-def*
apply(*simp-all add*: Δ*0-defs*)
by (*metis enat.distinct*(*2*) *f3 list.map-disc-iff llength-LNil ss3 zero-enat-def*)

show *match12-12* (*oor3* Δ*0* Δ*1* Δ*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* Δ*0 sstat* **unfolding** *ss cfg statA′* **apply** *simp*
apply(*simp add*: Δ*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*

show *oor3* Δ*0* Δ*1* Δ*1′* ∞ *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* Δ*0 stat* **unfolding** *ss* **apply**− **apply**(*frule* Δ*0-implies*)
by (*simp add*: Δ*0-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
show *?thesis* **using** *sa saO* Δ*0 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*: Δ*0-defs*) **by** (*metis config.sel*(*2*) *state.sel*(*2*))
subgoal apply(*subgoal-tac pc4* = *1*)
defer subgoal by (*simp add*: Δ*0-defs*)
subgoal using *xx-NN-cases*[*of vstore* (*getVstore* (*stateOf cfg3'*))] **apply**(*elim disjE*)
subgoal apply(*rule oor3I2*)
by (*simp add*: Δ*0-defs* Δ*1-defs*, *metis*)
subgoal apply(*rule oor3I3*)
apply (*simp add*: Δ*0-defs* Δ*1'-defs*)
apply(*intro conjI*, *metis*+)
apply *blast* **by** *fastforce*+
. . .
qed
qed
qed
qed
qed
qed

lemma *step1*: *unwindIntoCond* Δ*1* (*oor3* Δ*1* Δ*2* Δ*e*)
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*1*: Δ*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*:¬*finalN ss1*
  using Δ*1 finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB* Δ*1-defs*
  by *simp*

have *f2*:¬*finalN ss2*
  using Δ*1 finalB-pc-iff'* **unfolding** *ss cfg finalN-iff-finalB* Δ*1-defs*
  by *simp*

have *f3*:¬*finalS ss3*
  using Δ*1* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1-implies*)
  using *finalS-cond* **by** *simp*

have *f4*:¬*finalS ss4*
  using Δ*1* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1-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* Δ*1* Δ*2* Δ*e*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

  show *match1* (*oor3* Δ*1* Δ*2* Δ*e*) *ss3 ss4 statA ss1 ss2 statO*
  unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
  show *match2* (*oor3* Δ*1* Δ*2* Δ*e*) *ss3 ss4 statA ss1 ss2 statO*
  unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
  show *match12* (*oor3* Δ*1* Δ*2* Δ*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′ 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* Δ*1* **unfolding** *ss*
by (*simp add*: Δ*1-defs eqSec-def*)

show *eqSec ss2 ss4*
using *v sa* Δ*1* **unfolding** *ss*
by (*simp add*: Δ*1-defs eqSec-def*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1* **unfolding** *ss Van.eqAct-def*
by (*simp-all add*: Δ*1-defs*)

show *match12-12* (*oor3* Δ*1* Δ*2* Δ*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* Δ*1 sstat* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*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-all*
} **note** *stat* = *this*

show (*oor3* Δ*1* Δ*2* Δ*e*) ∞ *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* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1* *stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1* *stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1* *stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1* *stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1-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
      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: Δ1-defs)
            subgoal apply(rule oor3I1) by(simp add:Δ1-defs, metis)
            subgoal apply(rule oor3I1) by (simp add: Δ1-defs, metis)
            subgoal apply(rule oor3I1) by (simp add: Δ1-defs, metis)
            subgoal apply(rule oor3I1) by (simp add: Δ1-defs, metis)
            apply(rule oor3I3) by (simp-all add: Δ1-defs Δe-defs)
        qed
      qed
    qed
  qed
qed
```

qed

lemma *step2*: *unwindIntoCond* Δ*2* Δ*1*
proof(*rule unwindIntoCond-simpleI*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*2*: Δ*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*:¬*finalN ss1*
using Δ*2 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*2-defs*
by *simp*

have *f2*:¬*finalN ss2*
using Δ*2 finalB-pc-iff'* **unfolding** *ss finalN-iff-finalB* Δ*2-defs*
by *auto*

have *f3*:¬*finalS ss3*
using Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-implies*)
using *finalS-cond-spec* **by** *simp*

have *f4*:¬*finalS ss4*
using Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-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' 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 ¬ isSecO ss3
    using v sa Δ2 unfolding ss by (simp add: Δ2-defs)

    show ¬ isSecO ss4
    using v sa Δ2 unfolding ss apply clarsimp
    by (simp add: Δ2-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))

    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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 v3* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by**(*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2* **unfolding** *ss* **apply**−
apply(*frule* Δ*2-implies*) **by** (*simp add*: Δ*2-defs*)
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* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis* **using** *sa stat* Δ*2 v3 v4 sr3 sr4*
unfolding *ss lcfgs hh* **apply**−
by(*frule* Δ*2-implies*, *simp add*: Δ*2-defs* Δ*1-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* Δ*1′* Δ*3′* Δ*e*) *ss3 ss4 statA ss1 ss2 statO* ⟹
(∃ *v*<*n*. *proact* (*oor3* Δ*1′* Δ*3′* Δ*e*) *v ss3 ss4 statA ss1 ss2 statO*) ∨
react (*oor3* Δ*1′* Δ*3′* Δ*e*) *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* Δ*1′* (*oor3* Δ*1′* Δ*3′* Δ*e*)
proof(*rule unwindIntoCond-simpleIB*)
fix *n ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*1′*: Δ*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* = *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 Δ*1'*
  unfolding *ss* **apply**−**apply**(*frule* Δ*1'-implies*)
  unfolding *finalN-iff-finalB* Δ*1'-defs*
  using *finalB-pcOf-iff* **by** *simp*

have *f2*:¬*finalN ss2*
  using Δ*1'*
  unfolding *ss* **apply**−**apply**(*frule* Δ*1'-implies*)
  unfolding *finalN-iff-finalB* Δ*1'-defs*
  using *finalB-pcOf-iff* **by** *simp*

have *f3*:¬*finalS ss3*
  using Δ*1'* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1'-implies*)
  using *finalS-cond* **by** (*simp add*: Δ*1'-defs*)

have *f4*:¬*finalS ss4*
  using Δ*1'* **unfolding** *ss* **apply**−**apply**(*frule* Δ*1'-implies*)
  using *finalS-cond* **by** (*simp add*: Δ*1'-defs*)

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 (∃ *v*<*n*. *proact* (*oor3* Δ*1'* Δ*3'* Δ*e*) *v ss3 ss4 statA ss1 ss2 statO*) ∨
    react (*oor3* Δ*1'* Δ*3'* Δ*e*) *ss3 ss4 statA ss1 ss2 statO*
  using *cases-6*[*of pcOf cfg1*] **apply**(*elim disjE*)
  subgoal using Δ*1'* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*, *linarith*)
  subgoal using Δ*1'* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-less-numeral*)

show *eqSec ss2 ss4*
using *v sa* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-neq-numeral*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1′* **unfolding** *ss Van.eqAct-def*
apply (*simp-all add*: Δ*1′-defs*)
by (*metis* Δ*1′* Δ*1′-implies ss*)

show *match12-12* (*oor3* Δ*1′* Δ*3′* Δ*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** Δ*1′* **unfolding** *ss* Δ*1′-defs* **by** (*clarify*, *metis list.map-disc-iff*)

have *notJump*:¬*is-IfJump* (*prog* ! *pcOf cfg3*) **using** Δ*1′ pc* **unfolding** *ss* Δ*1′-defs*
by(*simp add*: Δ*1′-defs sstatO′-def sstatA′-def*)

{**assume** *sstat*: *statA′* = *Diff*
show *sstatO′ statO ss1 ss2* = *Diff*
using *v sa* Δ*1′ sstat pc* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*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* Δ*1′ pc* **unfolding** *ss cfg*
by (*simp-all add*: Δ*1′-defs*)

show (*oor3* Δ*1*′ Δ*3*′ Δ*e*) ∞ *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* Δ*1*′ *stat* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1*′ *stat* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1*′ *stat* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1*′ *stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1*′ *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1*′ *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1*′ *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1*′ *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1*′ *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1*′*-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
show *?thesis* **apply**(*rule oor3I1*)
using *sa* Δ*1*′ *stat pc pc4 v3 v4 nn3 config.sel*(*2*) *state.sel*(*2*)

unfolding *ss cfg cfg1 cfg2 hh* **apply**(*simp add*:Δ*1′-defs*)
using *numeral-le-iff semiring-norm*(*69*,*72*) **by** *force*
qed
qed
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′ 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* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-less-numeral*)

show *eqSec ss2 ss4*
using *v sa* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-neq-numeral*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1′* **unfolding** *ss Van.eqAct-def*
apply (*simp-all add*: Δ*1′-defs*)
by (*metis* Δ*1′* Δ*1′-implies ss*)

show *match12-12* (*oor3* Δ*1′* Δ*3′* Δ*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** Δ*1′* **unfolding** *ss* Δ*1′-defs* **by** (*clarify*,*metis map-is-Nil-conv*)

{**assume** *sstat*: *statA′* = *Diff*
show *sstatO′ statO ss1 ss2* = *Diff*

```
using v sa Δ1' sstat pc unfolding ss cfg statA'
apply(simp add: Δ1'-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 Δ1' pc unfolding ss cfg
 by (simp-all add: Δ1'-defs)


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

  case spec-normal
  then show ?thesis using sa Δ1' stat nm3 unfolding ss by (simp add:
Δ1'-defs cfgs4)
 next
  case spec-mispred
  then show ?thesis using sa Δ1' stat nm3 unfolding ss by (simp add:
Δ1'-defs cfgs4)
 next
  case spec-Fence
  then show ?thesis using sa Δ1' stat nm3 unfolding ss by (simp add:
Δ1'-defs cfgs4)
 next
  case spec-resolve
  then show ?thesis using sa Δ1' stat nm3 unfolding ss by (simp add:
Δ1'-defs cfgs4)
```

next
case *nonspec-normal*
then show *?thesis* **using** *sa* Δ*1′ stat nm3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs cfgs4*)
next
case *nonspec-mispred* **note** *nm4* = *nonspec-mispred*
show *?thesis* **apply**(*rule oor3I2*)
using *sa pc4* Δ*1′ stat pc v3 v4 nm3 nm4 config.sel*(*2*) *state.sel*(*2*)
unfolding *ss cfg cfg1 cfg2 hh* **apply**(*simp add*:Δ*1′-defs* Δ*3′-defs*)
by (*metis empty-subsetI nat-less-le nat-neq-iff numeral-eq-iff semiring-norm*(*89*) *set-eq-subset*)
qed
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* Δ*1′ stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1′ stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1′ stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1′ stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1′ stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1′-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
show *?thesis* **apply**(*rule oor3I1*)
using *sa pc4* Δ*1′ stat pc v3 v4 nn3 config.sel*(*2*) *state.sel*(*2*)
unfolding *ss cfg cfg1 cfg2 hh* **apply**(*simp add*:Δ*1′-defs*)
by (*metis nat-le-linear nat-less-le numeral-eq-iff semiring-norm*(*88*))
qed
qed
qed
qed
subgoal apply(*rule disjI1*, *rule exI*[*of* - *2*], *rule conjI*)
subgoal using Δ*1′* **unfolding** *ss* Δ*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 ¬ *isSecV ss1* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg* **by** *auto*

show ¬ *isSecV ss2* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg* **by** *auto*

show *Van.eqAct ss1 ss2* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg Van.eqAct-def* **by** *auto*

show *move-12* (*oor3* Δ*1′* Δ*3′* Δ*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 Δ*1′ pc* **unfolding** *validTransV-iff-nextN ss* Δ*1′-defs*
by *simp*

show *validTransV* (*ss2*, *nextN ss2*)
using Δ*1′ pc* **unfolding** *validTransV-iff-nextN ss* Δ*1′-defs*
by *simp*
have *a1-0*:*array-loc aa1 0 avst3* = *array-loc aa1 0 avst4*
using Δ*1′ pc* **unfolding** *cfg cfg1 ss* Δ*1′-defs array-loc-def* **by** *simp*
have *pc1*:*pc1* = *4* **using** Δ*1′ pc* **unfolding** *cfg cfg1 ss* Δ*1′-defs* **by** *simp*

show *oor3* Δ*1′* Δ*3′* Δ*e 2 ss3 ss4 statA* (*nextN ss1*) (*nextN ss2*) (*sstatO′ statO ss1 ss2*)
apply(*rule oor3I1*)
using Δ*1′ pc* **unfolding** *ss cfg cfg1 cfg2 hh h1 h2 endPC* **apply**(*simp add*: Δ*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 Δ*1′* **unfolding** *ss* Δ*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 ¬ *isSecV ss1* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg* **by** *auto*

show ¬ *isSecV ss2* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg* **by** *auto*

show *Van.eqAct ss1 ss2* **using** Δ*1′ pc* **unfolding** Δ*1′-defs ss cfg Van.eqAct-def* **by** *auto*

show *move-12* (*oor3* Δ*1′* Δ*3′* Δ*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 Δ*1′ pc* **unfolding** *validTransV-iff-nextN ss* Δ*1′-defs*
by *simp*

show *validTransV* (*ss2*, *nextN ss2*)
using Δ*1′ pc* **unfolding** *validTransV-iff-nextN ss* Δ*1′-defs*
by *simp*

show *oor3* Δ*1′* Δ*3′* Δ*e 1 ss3 ss4 statA* (*nextN ss1*) (*nextN ss2*) (*sstatO′ statO ss1 ss2*)
apply(*rule oor3I1*)
using Δ*1′ pc* **unfolding** *ss cfg cfg1 cfg2 hh h1 h2 endPC* **apply**(*simp add*: Δ*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′ 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* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-less-numeral*)

show *eqSec ss2 ss4*
using *v sa* Δ*1′* **unfolding** *ss* **apply** (*simp add*: Δ*1′-defs*)
by (*metis not-gr-zero not-numeral-le-zero zero-neq-numeral*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1′* **unfolding** *ss Van.eqAct-def*
apply (*simp-all add*: Δ*1′-defs*)
by (*metis* Δ*1′* Δ*1′-implies ss*)

show *match12-12* (*oor3* Δ*1′* Δ*3′* Δ*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** Δ*1′* **unfolding** *ss* Δ*1′-defs* **by** (*clarify*,*metis map-is-Nil-conv*)

{**assume** *sstat*: *statA′* = *Diff*
show *sstatO′ statO ss1 ss2* = *Diff*
using *v sa* Δ*1′ sstat pc* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*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* Δ*1′ pc* **unfolding** *ss cfg*
by (*simp-all add*: Δ*1′-defs*)

have *notJump*:¬*is-IfJump* (*prog* ! *pcOf cfg3*) **using** Δ*1′ pc* **unfolding** *ss* Δ*1′-defs*
by(*simp add*: Δ*1′-defs sstatO′-def sstatA′-def*)

show (*oor3* Δ*1′* Δ*3′* Δ*e*) ∞ *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* Δ*1'* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1'* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1'* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1'* *stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1'* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1'* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1'* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1'* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1'* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
show *?thesis* **apply**(*rule oor3I3*)
using *sa* Δ*1'* *stat pc pc4 v3 v4 nn3 config.sel*(*2*) *state.sel*(*2*)
unfolding *ss cfg cfg1 cfg2 hh* **by**(*simp add*:Δ*1'-defs* Δ*e-defs*)

qed
qed
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 Δ*3′* **unfolding** *ss* **apply** (*simp add*: Δ*3′-defs*)
  by (*metis length-1-butlast length-map*)

have *h3-eq*:*hh3* = *lhh3*
  using *cfg lcfgs hh* Δ*3′* **unfolding** Δ*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* Δ*3′* **unfolding** Δ*3′-defs ss* **apply** *clarify*
  using *config.sel*(*2*) *getHheap.simps heap.sel last-in-set*
  by (*metis map-is-Nil-conv*)

have *f1*:¬*finalN ss1*
  using Δ*3′ finalB-pc-iff′* **unfolding** *ss finalN-iff-finalB* Δ*3′-defs*
  by *simp*

have *f2*:¬*finalN ss2*
  using Δ*3′ finalB-pc-iff′* **unfolding** *ss cfg finalN-iff-finalB* Δ*3′-defs*
  by *simp*

have *f3*:¬*finalS ss3*
  using Δ*3′* **unfolding** *ss* **apply**−**apply**(*frule* Δ*3′-implies*)
  using *finalS-cond-spec* **by** (*simp add*: Δ*3′-defs*)

have *f4*:¬*finalS ss4*
  using Δ*3′* **unfolding** *ss* **apply**−**apply**(*frule* Δ*3′-implies*)
  using *finalS-cond-spec* **apply** (*simp add*: Δ*3′-defs*)
  by (*metis length-map*)

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* Δ*3′* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*

unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* Δ*3′* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor* Δ*3′* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor* Δ*3′* Δ*1′*) *ss3 ss4 statA ss1 ss2 statO*
using *cases-thenBranch*[*of pcOf* (*last cfgs3*)]
apply(*elim disjE*)
subgoal using Δ*3′* **unfolding** *ss lcfgs* Δ*3′-defs*
by (*clarify*, *metis atLeastAtMost-iff inThenBranch-def lcfgs3 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 cfgs3*) = *4*
note *v3* = *v*(*1*) **note** *v4* = *v*(*2*)

have *pc2*:*pc2* = *4*
using Δ*3′ pc* **unfolding** *ss cfg* **unfolding** Δ*3′-defs*
apply *clarify*
by (*metis 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* Δ*3′* **unfolding** *ss* **by** (*simp add*: Δ*3′-defs*)

show *eqSec ss2 ss4*
using *v sa* Δ*3′* **unfolding** *ss* **by** (*simp add*: Δ*3′-defs*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*3′* **unfolding** *ss Van.eqAct-def*

```
      by (simp add: Δ3'-defs lessI less-or-eq-imp-le numeral-3-eq-3 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 finals ss status.distinct(1) new-
Stat.simps(1))
      } note stat = this


        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
         then show ?thesis using sa Δ3' stat pc unfolding ss apply (simp add:
Δ3'-defs)
         by (metis last-ConsL last-map n-not-Suc-n numeral-2-eq-2 numeral-3-eq-3
numeral-eq-iff semiring-norm(87))

        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 Δ3' stat sn3 unfolding ss by (simp add:
Δ3'-defs)
          next
            case spec-mispred
           then show ?thesis using sa Δ3' stat sn3 unfolding ss by (simp add:
Δ3'-defs)
          next
            case spec-Fence
           then show ?thesis using sa Δ3' stat sn3 unfolding ss by (simp add:
Δ3'-defs)
          next
            case spec-resolve
           then show ?thesis using sa Δ3' stat sn3 unfolding ss by (simp add:
Δ3'-defs)
          next
            case nonspec-normal note nn4 = nonspec-normal
           then show ?thesis using sa Δ3' stat sn3 unfolding ss by (simp add:
Δ3'-defs)
          next
            case spec-normal note sn4 = spec-normal
            then show ?thesis
              using Δ3' 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: Δ3'-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
        qed
      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* Δ*3′* **unfolding** *ss* **by** (*simp add*: Δ*3′-defs pc*)

show *eqSec ss2 ss4*
using *v sa* Δ*3′* **unfolding** *ss* **by** (*simp add*: Δ*3′-defs pc*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*3′* **unfolding** *ss Van.eqAct-def*
by (*simp add*: Δ*3′-defs pc*)

show *match12-12* (*oor* Δ*3′* Δ*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* Δ*3′ sstat* **unfolding** *ss cfg statA′*
apply(*simp add*: Δ*3′-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*) *newStat.simps*(*1*))
} **note** *stat* = *this*

show *oor* Δ*3′* Δ*1′* ∞ *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* Δ*3′ stat* **unfolding** *ss* **by** (*simp add*: Δ*3′-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*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*: $\Delta 3'$-*defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* $\Delta 3'$ *stat pc* **unfolding** *ss* **apply** (*simp add*: $\Delta 3'$-*defs*)
by (*metis last-ConsL last-map numeral-eq-iff semiring-norm*(*89*))

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 lcfgs 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
qed

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 ¬ *isSecO ss3*
using *v sa* Δ*3'* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)

show ¬ *isSecO ss4*
using *v sa* Δ*3'* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)

show *stat*: *statA* = *statA'* ∨ *statO* = *Diff*
using *v sa* Δ*3'*
unfolding *ss statA' sstatA'-def*
apply(*simp-all add*: Δ*3'-defs*)
apply (*cases statA*, *simp-all*)
by (*smt* (*verit*, *best*) *Nil-is-map-conv f3 f4 ss newStat.simps*(*1*))

show *oor* Δ*3'* Δ*1'* ∞ *ss3' ss4' statA' ss1 ss2 statO*
using *v3*[*unfolded ss*, *simplified*] **proof**(*cases rule*: *stepS-cases*)
case *nonspec-mispred*
then show *?thesis* **using** *sa* Δ*3' stat* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*3' stat* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*3' stat* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)
next
case *nonspec-normal*
then show *?thesis* **using** *sa* Δ*3' stat* **unfolding** *ss* **by** (*simp add*: Δ*3'-defs*)

next
case *spec-normal* **note** *sn3* = *spec-normal*
show *?thesis* **using** *sa* $\Delta 3'$ *stat sn3 pc v3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-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* $\Delta 3'$ *stat sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* $\Delta 3'$ *stat sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* $\Delta 3'$ *stat sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-defs*)
next
case *nonspec-normal*
then show *?thesis* **using** *sa* $\Delta 3'$ *stat sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* $\Delta 3'$ *stat sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 3'$*-defs*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
then show *?thesis*
using $\Delta 3'$ *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*: $\Delta 3'$*-defs* $\Delta 1'$*-defs*, *intro conjI*, *metis*)
apply *meson* **apply** *meson* **apply** *blast* **by** *meson*
qed
qed
qed
subgoal using $\Delta 3'$ **unfolding** *ss lcfgs* $\Delta 3'$*-defs*
by (*simp add*: *avstoreOf.cases elseBranch-def lcfgs3*) **.**
qed
qed

lemma *stepe*: *unwindIntoCond* Δe Δe
proof(*rule unwindIntoCond-simpleI*)

```
  fix n ss3 ss4 statA ss1 ss2 statO
  assume r: reachO ss3 reachO ss4 reachV ss1 reachV ss2
    and Δe: Δe 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 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
    using Δe Opt.final-def Prog.endPC-def finalS-def stepS-endPC
    unfolding Δe-defs ss by clarsimp


  then show isIntO ss3 = isIntO ss4 by simp

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

    show match1 Δe ss3 ss4 statA ss1 ss2 statO
      unfolding match1-def by (simp add: finalS-def final-def)
    show match2 Δe 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 Δ*s* **where** Δ*s*: Δ*s* ≡ λ*i*::*nat*.
if i = 0 then Δ*0*
else if i = 1 then Δ*1*
else if i = 2 then Δ*2*
else if i = 3 then Δ*1'*
else if i = 4 then Δ*3'*
else Δ*e*
define *nxt* **where** *nxt*: *nxt* ≡ λ*i*::*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* Δ*s*])
subgoal unfolding *m* **by** *auto*
subgoal unfolding *nxt m* **by** *auto*
subgoal using *init* **unfolding** Δ*s* **by** *auto*
subgoal
unfolding *m nxt* Δ*s* **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* (‹⊥›)
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* ⇒ *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* ≡
[
0 Start ,
1 tt ::= (N 0),
2 xx ::= (N 1),
3 IfJump (Not (Eq (V xx) (N 0))) 4 11 ,
4 Input U xx ,
5 IfJump (Less (V xx) (N NN)) 6 10 ,
6 vv ::= VA aa1 (V xx) ,
7 Fence ,
8 tt ::= (VA aa2 (Times (V vv) (N SS))) ,
9 Output U (V tt) ,
10 Jump 3,
11 Output U (N 0)
]

definition *PC* ≡ {*0..11*}

definition *beforeWhile* = {*0,1,2*}
definition *inWhile* = {*3..11*}
definition *startOfWhileThen* = *4*

definition *startOfIfThen* = *6*
definition *inThenIfBeforeFence* = {*6*,*7*}
definition *startOfElseBranch* = *10*
definition *inElseIf* = {*10*,*3*,*4*,*11*}
definition *whileElse* = *11*

fun *leftWhileSpec* **where**
leftWhileSpec cfg cfg′ =
(*pcOf cfg* = *whileElse* ∧
pcOf cfg′ = *startOfWhileThen*)

fun *rightWhileSpec* **where**
rightWhileSpec cfg cfg′ =
(*pcOf cfg* = *startOfWhileThen* ∧
pcOf cfg′ = *whileElse*)

fun *whileSpeculation* **where**
whileSpeculation cfg cfg′ =
(*leftWhileSpec cfg cfg′* ∨
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* ∨ *i* = *1* ∨ *i* = *2* ∨ *i* = *3* ∨ *i* = *4* ∨ *i* = *5* ∨ *i* = *6* ∨ *i* = *7* ∨ *i* = *8* ∨ *i* = *9* ∨ *i* = *10* ∨ *i* = *11* ∨ *i* = *12* ∨ *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* ∨ *vs xx* ≠ *0* **by** *auto*

lemma *xx-NN-cases*: *vs xx* < *int NN* ∨ *vs xx* ≥ *int NN* **by** *auto*

lemma *is-IfJump-pcOf*[*simp*]:
pcOf cfg < *12* ⟹ *is-IfJump* (*prog* ! (*pcOf cfg*)) ⟷ *pcOf cfg* = *3* ∨ *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* ⟹ *is-IfJump* (*prog* ! *pc*) ⟷ *pc* = *3* ∨ *pc* = *5*
using *cases-12*[*of pc*]
by (*auto simp*: *prog-def*)

lemma *eq-Fence-pc*[*simp*]:
pc < *12* ⟹ *prog* ! *pc* = *Fence* ⟷ *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*]:¬*is-IfJump* (*prog* ! *6*) **by**(*simp add*: *prog-def*)
lemma *is-nif2*[*simp*]:¬*is-IfJump* (*prog* ! *7*) **by**(*simp add*: *prog-def*)

lemma *is-nin1*[*simp*]:¬ *is-getInput* (*prog* ! *6*) **by**(*simp add*: *prog-def*)
lemma *is-nout1*[*simp*]:¬ *is-Output* (*prog* ! *6*) **by**(*simp add*: *prog-def*)
lemma *is-nin2*[*simp*]:¬ *is-getInput* (*prog* ! *10*) **by**(*simp add*: *prog-def*)
lemma *is-nout2*[*simp*]:¬ *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* ≠ *Fence* **by**(*simp add*: *prog-def*)

consts *mispred* :: *predState* ⇒ *pcounter list* ⇒ *bool*
fun *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool* **where**
resolve p pc =
(*if* (*set pc* = {*4*,*11*} ∨ (*6* ∈ *set pc* ∧ (*4* ∈ *set pc* ∨ *11* ∈ *set pc*)))
then True else False)

lemma *resolve-63*: ¬*resolve p* [*6*,*3*] **by** *auto*
lemma *resolve-64*: *resolve p* [*6*,*4*] **by** *auto*
lemma *resolve-611*: *resolve p* [*6*,*11*] **by** *auto*
lemma *resolve-106*: ¬*resolve p* [*10*,*6*] **by** *auto*

consts *update* :: *predState* ⇒ *pcounter list* ⇒ *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* ⟹ *is-getInput* (*prog*!(*pcOf cfg*)) ⟷ *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*]: ¬*is-getInput* (*prog* ! *6*) **by** (*auto simp*: *prog-def*)
lemma *getInput-not7*[*simp*]: ¬*is-getInput* (*prog* ! *7*) **by** (*auto simp*: *prog-def*)
lemma *getInput-not10*[*simp*]: ¬*is-getInput* (*prog* ! *10*) **by** (*auto simp*: *prog-def*)

lemma *is-Output-pcOf*[*simp*]: *pcOf cfg* < *12* ⟹ *is-Output* (*prog*!(*pcOf cfg*)) ⟷ (*pcOf cfg* = *9* ∨ *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*) ⟷ *pcOf cfg* = *0*
using *isSecV-def* **by** *simp*

lemma *isSecO-pcOf*[*simp*]:
isSecO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟷ (*pcOf cfg* = *0* ∧ *cfgs* = [])
using *isSecO-def* **by** *simp*

lemma *getInputT-not*[*simp*]: *pcOf cfg* < *12* ⟹
(*prog* ! *pcOf cfg*) ≠ *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* ⟹
getActV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *4 then lhd ibUT else* ⊥)
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* ⟹
getObsV (*cfg*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *9* ∨ *pcOf cfg* = *11 then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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* ⟹
getActO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if pcOf cfg* = *4* ∧ *cfgs* = [] *then lhd ibUT else* ⊥)
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* ⟹
getObsO (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) =
(*if* (*pcOf cfg* = *9* ∨ *pcOf cfg* = *11*) ∧ *cfgs* = [] *then*
(*outOf* (*prog*!(*pcOf cfg*)) (*stateOf cfg*), *ls*)
else ⊥
)
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*) ⟷
(*pcOf cfg1* = *0* ⟷ *pcOf cfg3* = *0* ∧ *cfgs3* = []) ∧
(*pcOf cfg1* = *0* ⟶ *stateOf cfg1* = *stateOf cfg3*)
unfolding *eqSec-def* **by** *simp*

lemma *getActInput*:*pc4* = *4* ⟹ *pc3* = *4* ⟹ *cfgs3* = [] ⟹ *cfgs4* = [] ⟹
getActO (*pstate3*, *Config pc3* (*State* (*Vstore vs3*) *avst3 h3 p3*), [], *ibT3*,*ibUT3*, *ls3*) =
getActO (*pstate4*, *Config pc4* (*State* (*Vstore vs4*) *avst4 h4 p4*), [], *ibT4*,*ibUT4*, *ls4*)
⟹ *lhd ibUT3* = *lhd ibUT4*

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 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* ⟹
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* ⟹
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* ⟹ *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* ⟹
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 ≥ *int NN* ⟹
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 ≥ *int NN* ⟹
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* ⟹
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* ⟹
  pcOf (*last cfgs*) = *y* ⟹ *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* ⟹
  pcOf (*cfgs* ! *0*) = *x* ⟹
  pcOf (*last cfgs*) = *y* ⟹ *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 *length cfgs* = *2* ⟹ (*cfgs* ! *0*) = *last* (*butlast cfgs*)
 by (*cases cfgs*, *auto*)

lemma *nextB-stepB-pc*:
pc < *12* ⟹ (*pc* = *4* ⟶ *ibUT* ≠ *LNil*) ⟹
(*Config pc s*, *ibT*,*ibUT*) →*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 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*:
$pc < 12 \Longrightarrow (pc = 4 \longrightarrow ibUT \neq LNil) \Longrightarrow$
$\neg$ *finalB* (*Config pc s*, *ibT*,*ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff'*:
$pc < 12 \Longrightarrow$
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*:
$pc \leq 12 \Longrightarrow$
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*]:
$pcOf\ cfg \leq 12 \Longrightarrow$
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* ⟹ *noMisSpec cfgs* ⟹ *ibUT* ≠ *LNil* ⟹ ¬ *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* ∪ *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* ∪ *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* ∪ *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*)
pstate update pstate [*pcOf* (*Config pc* (*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 Config (*if vs xx* ≠ *0 then 11 else 4*) (*State* (*Vstore vs*) *avst hh p*)
ibT ibUT [*Config* (*if vs xx* ≠ *0 then 11 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 cfgs Config pc* (*State* (*Vstore vs*) *avst*

hh p)
pstate pstate ibT ibUT
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*)) *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 5 (*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(*cases mispred pstate* [*5*])
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 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 ∪ *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 6 else 10*) (*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 xx*)) (*Avstore as*))
in (*Config 7* (*State* (*Vstore* (*vs*(*vv* := *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 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 using *stepS-spec-resolve-iff*[*of cfgs pstate cfg ibT ibUT ls update pstate* (*pcOf cfg # map pcOf cfgs*)]
by (*metis* (*no-types*, *lifting*) *cfgs-map empty-set insert-commute less-numeral-extra*(*3*)

resolve.simps list.simps(*15*) *list.size*(*3*) *numeral-2-eq-2 pos2*)
subgoal apply(*elim conjE*)
using *spec-resolve*[*of cfgs pstate cfg update pstate* (*pcOf cfg # map pcOf cfgs*) *cfg* [] *ibT ibT ibUT ibUT ls ls*]
by (*metis* (*no-types*, *lifting*) *empty-set insert-commute last-ConsL resolve.simps*
length-0-conv length-1-butlast length-Suc-conv list.simps(*9*,*8*,*15*)) **. .**
. .

lemma *finalS-while-spec-L2*:
pcOf cfg = *6* ⟹
whileSpeculation (*cfgs*!*0*) (*last cfgs*) ⟹
length cfgs = *2* ⟹
¬ *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 using *stepS-spec-resolve-iff*[*of cfgs pstate cfg ibT ibUT ls update pstate* (*pcOf cfg # map pcOf cfgs*)]
unfolding *resolve.simps*
using *list.set-intros*(*1*,*2*) *map-L2 zero-neq-numeral*
by *fastforce*
subgoal apply(*elim conjE*)
using *spec-resolve*
unfolding *resolve.simps*
using *list.set-intros*(*1*,*2*) *map-L2 zero-neq-numeral*
by (*metis* (*no-types*, *lifting*) *Prog-Mispred.spec-resolve Prog-Mispred-axioms list.size*(*3*))
. . . .

lemma *finalS-if-spec*:
(*pcOf* (*last cfgs*) ∈ *inThenIfBeforeFence* ∧ *pcOf cfg* = *10*) ∨
(*pcOf* (*last cfgs*) ∈ *inElseIf* ∧ *pcOf cfg* = *6*) ⟹
length cfgs = *Suc 0* ⟹
¬ *finalS* (*pstate*, *cfg*, *cfgs*, *ibT*,*ibUT*, *ls*)
unfolding *inThenIfBeforeFence-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(*rule notI*,
erule allE[*of* - (*pstate*,*cfg*,
[*Config 7* (*State* (*Vstore* (*vs*(*vv* := *h* (*array-loc aa1* (*nat* (*vs xx*)) *avst*)))) *avst hh p*)],
ibT,*ibUT*,*ls* ∪ *readLocs* (*last cfgs*))])
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 cfgs-Suc-zero fence not-Cons-self2 stepS-spec-Fence-iff spec-resolve*)
subgoal apply(*elim conjE*, *elim disjE*)
subgoal apply(*rule notI*,
erule allE[*of* - (*pstate*,*cfg*,
[*Config 3* (*State* (*Vstore vs*) *avst hh p*)],
ibT,*ibUT*,*ls* ∪ *readLocs* (*last cfgs*))])
by(*erule notE*, *rule spec-normal*[*of* - - - - - -*Config 3* (*State* (*Vstore vs*) *avst hh p*)], *auto*)
subgoal apply(*cases mispred pstate* [*6*,*3*])
subgoal apply(*rule notI*, *erule allE*[*of* -
(*update pstate* (*pcOf cfg* # *map pcOf cfgs*),
cfg,
[*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*, *cfg*, [*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 by (*metis resolve-64 stepS-spec-resolve-iff*
map-L1 cfgs-Suc-zero not-Cons-self2)
subgoal by (*metis resolve-611 stepS-spec-resolve-iff*
map-L1 cfgs-Suc-zero not-Cons-self2)
. . . . .

end

12.2 Proof

theory *Fun5-secure*
imports *Fun5*
begin

definition *common* :: *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool*
where
common = (λ*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* ∧
cfg1 = *cfg3* ∧ *cfg2* = *cfg4* ∧
pcOf cfg3 = *pcOf cfg4* ∧ *map pcOf cfgs3* = *map pcOf cfgs4* ∧
pcOf cfg3 ∈ *PC* ∧ *pcOf* ‘ (*set cfgs3*) ⊆ *PC* ∧
llength ibUT1 = ∞ ∧ *llength ibUT2* = ∞ ∧
ibUT1 = *ibUT3* ∧ *ibUT2* = *ibUT4* ∧

w1 = *w2* ∧
///
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*))) ∧
(∀ *cfg4'*∈*set cfgs4*. *array-base aa2* (*getAvstore* (*stateOf cfg4'*)) = *array-base aa2* (*getAvstore* (*stateOf cfg4*))) ∧
///
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
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 ⟹
pcOf cfg1 < *12* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
ibUT1 ≠ [[]] ∧ *ibUT2* ≠ [[]] ∧ *w1* = *w2*
unfolding *common-def PC-def* **by** (*auto simp*: *image-def subset-eq*)

definition Δ*0* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*0* = (λ*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 ∧
pcOf cfg3 ∈ *beforeWhile* ∧
(*pcOf cfg3* > *1* ⟶ *same-var-o tt cfg3 cfgs3 cfg4 cfgs4*) ∧
(*pcOf cfg3* > *2* ⟶ *same-var-o xx cfg3 cfgs3 cfg4 cfgs4*) ∧
(*pcOf cfg3* > *4* ⟶ *same-var-o xx cfg3 cfgs3 cfg4 cfgs4*) ∧
noMisSpec cfgs3
))

lemmas Δ*0-defs* = Δ*0-def common-def PC-def same-var-o-def*
beforeWhile-def noMisSpec-def

lemma Δ*0-implies*: Δ*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 ⟹
pcOf cfg1 < *12* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
ibUT1 ≠ [[]] ∧ *ibUT2* ≠ [[]] ∧ *cfgs4* = []
apply (*meson* Δ*0-def common-implies*)
by (*simp-all add*: Δ*0-defs*, *metis Nil-is-map-conv*)

definition Δ*1* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*1* = (λ *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 ∧
pcOf cfg3 ∈ *inWhile* ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
noMisSpec cfgs3
))
lemmas Δ*1-defs* = Δ*1-def common-def PC-def noMisSpec-def inWhile-def same-var-o-def*
lemma Δ*1-implies*: Δ*1 n w1 w2* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf cfg3 < *12* ∧ *cfgs3* = [] ∧ *ibUT3* ≠ [[]] ∧
pcOf cfg4 < *12* ∧ *cfgs4* = [] ∧ *ibUT4* ≠ [[]]
unfolding Δ*1-defs* **apply** *simp*
by (*metis Nil-is-map-conv infinity-ne-i0 llength-LNil*)

definition Δ*1′* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*1′* = (λ*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 ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
whileSpeculation cfg3 (*last cfgs3*) ∧
misSpecL1 cfgs3 ∧ *misSpecL1 cfgs4* ∧
w1 = ∞
))
lemmas Δ*1′-defs* = Δ*1′-def common-def PC-def same-var-def*
startOfIfThen-def startOfElseBranch-def
misSpecL1-def whileSpec-defs

lemma Δ*1′-implies*: Δ*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 ⟹
pcOf cfg3 < *12* ∧ *pcOf cfg4* < *12* ∧
whileSpeculation cfg3 (*last cfgs3*) ∧
whileSpeculation cfg4 (*last cfgs4*) ∧
length cfgs3 = *Suc 0* ∧ *length cfgs4* = *Suc 0*
unfolding Δ*1′-defs*
apply (*simp add*: *lessI*, *clarify*)
by (*metis last-map length-0-conv*)

definition Δ*2* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*2* = (λ*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 ∧

same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfIfThen* ∧ *pcOf* (*last cfgs3*) ∈ *inElseIf* ∧
misSpecL1 cfgs3 ∧ *misSpecL1 cfgs4* ∧

(*pcOf* (*last cfgs3*) = *startOfElseBranch* ⟶ *w1* = ∞) ∧
(*pcOf* (*last cfgs3*) = *3* ⟶ *w1* = *3*) ∧

(*pcOf* (*last cfgs3*) = *startOfWhileThen* ∨
pcOf (*last cfgs3*) = *whileElse* ⟶ *w1* = *1*)
))

lemmas Δ*2-defs* = Δ*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 Δ*2-implies*: Δ*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 ⟹
pcOf (*last cfgs3*) ∈ *inElseIf* ∧ *pcOf cfg3* = *6* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg4 = *pcOf cfg3* ∧ *length cfgs3* = *Suc 0* ∧
length cfgs4 = *Suc 0* ∧ *same-var xx* (*last cfgs3*) (*last cfgs4*)
apply(*intro conjI*)
  unfolding Δ*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 Δ*2′* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*2′* = (λ*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 ∧
  same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
  pcOf cfg3 = *startOfIfThen* ∧
  whileSpeculation (*cfgs3*!*0*) (*last cfgs3*) ∧
  misSpecL2 cfgs3 ∧ *misSpecL2 cfgs4* ∧
  w1 = *2*
))

lemmas Δ*2′-defs* = Δ*2′-def common-def PC-def same-var-def*
    startOfElseBranch-def startOfIfThen-def
    whileSpec-defs misSpecL2-def

lemma Δ*2′-implies*: Δ*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 ⟹
 pcOf cfg3 = *6* ∧ *pcOf cfg4* = *6* ∧
 whileSpeculation (*cfgs3*!*0*) (*last cfgs3*) ∧
 whileSpeculation (*cfgs4*!*0*) (*last cfgs4*) ∧
 length cfgs3 = *2* ∧ *length cfgs4* = *2*
 apply(*intro conjI*)
 unfolding Δ*2′-defs* **apply** (*simp add*: *lessI*, *clarify*)
 apply *linarith*+ **apply** *simp-all*

by (*metis list.inject map-L2*)

definition $\Delta 3$ :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 3$ = (λ*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 ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfElseBranch* ∧ *pcOf* (*last cfgs3*) ∈ *inThenIfBeforeFence* ∧
misSpecL1 cfgs3 ∧
(*pcOf* (*last cfgs3*) = *6* ⟶ *w1* = ∞) ∧
(*pcOf* (*last cfgs3*) = *7* ⟶ *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*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf (*last cfgs3*) ∈ *inThenIfBeforeFence* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *10* ∧ *pcOf cfg3* = *pcOf cfg4* ∧
length cfgs3 = *Suc 0* ∧ *length cfgs4* = *Suc 0*
apply(*intro conjI*)
unfolding $\Delta 3$*-defs*
apply (*simp-all add*: *image-subset-iff*)
by (*metis last-map map-is-Nil-conv length-map*)+

definition Δe :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δe = (λ*num w1 w2* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)

statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO.
(*pcOf cfg3* = *endPC* ∧ *pcOf cfg4* = *endPC* ∧ *cfgs3* = [] ∧ *cfgs4* = [] ∧
pcOf cfg1 = *endPC* ∧ *pcOf cfg2* = *endPC*))

lemmas Δ*e-defs* = Δ*e-def common-def endPC-def*

lemma *init*: *initCond* Δ*0*
unfolding *initCond-def* **apply** *safe*
subgoal for *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3 pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4*

unfolding *istateO.simps* **apply** *clarsimp*
apply(*cases getAvstore* (*stateOf cfg3*), *cases getAvstore* (*stateOf cfg4*))
unfolding Δ*0-defs*
unfolding *array-base-def* **by** *auto* **.**

lemma *step0*: *unwindIntoCond* Δ*0* (*oor* Δ*0* Δ*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 Δ*0*: Δ*0 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 Δ*0* **unfolding** *ss*
apply−**by**(*frule* Δ*0-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*0* **unfolding** *ss*
apply−**by**(*frule* Δ*0-implies*, *simp*)

have *f3*:¬*finalS ss3*
using Δ*0* **unfolding** *ss*
apply−**apply**(*frule* Δ*0-implies*, *unfold* Δ*0-defs*)
by (*clarify*,*metis finalS-cond′*)

have *f4*:¬*finalS ss4*
using Δ*0* **unfolding** *ss*
apply−**apply**(*frule* Δ*0-implies*, *unfold* Δ*0-defs*)
by (*clarify*,*metis finalS-cond′*)

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* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor* Δ*0* Δ*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 = 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 unfolding ss by (simp add: Δ0-defs)

show eqSec ss2 ss4
using v sa Δ0  unfolding ss
apply (simp add: Δ0-defs)
by (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-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

 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
        by (simp add: Δ0-defs numeral-1-eq-Suc-0, linarith)
      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 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 Δ0 stat nn3 unfolding ss by (simp add:
Δ0-defs)
        next
          case spec-normal
           then show ?thesis using sa Δ0 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 Δ0 stat v3 v4 nn3 nn4 unfolding ss cfg apply
clarsimp
          apply(unfold Δ0-defs, clarsimp, elim disjE)
            subgoal by(rule oorI1, auto simp add: Δ0-defs)
            subgoal by (rule oorI1, simp add: Δ0-defs)
            subgoal by (rule oorI2, simp add: Δ1-defs) .
        qed
      qed
```

```
      qed
    qed
  qed
qed


lemma step1: unwindIntoCond Δ1 (oor5 Δ1 Δ1' Δ2 Δ3 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 Δ1: Δ1 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 Δ1 unfolding ss Δ1-def apply clarify
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by blast

  have f2:¬finalN ss2
  using Δ1 unfolding ss Δ1-def apply clarify
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by metis


  have f3:¬finalS ss3
```

using $\Delta 1$ **unfolding** *ss*
apply−**apply**(*frule* Δ*1-implies*)
by (*simp add*: *finalS-cond'*)

have *f4*:¬*finalS ss4*
using $\Delta 1$ **unfolding** *ss*
apply−**apply**(*frule* Δ*1-implies*)
by (*simp add*: *finalS-cond'*)

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* (*oor5* $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor5* $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor5* $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor5* $\Delta 1$ $\Delta 1'$ $\Delta 2$ $\Delta 3$ Δe) *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'*

show *eqSec ss1 ss3*
using *v sa* $\Delta 1$ **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

show *eqSec ss2 ss4*
using *v sa* $\Delta 1$ **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

show *Van.eqAct ss1 ss2*
using *v sa* $\Delta 1$ **unfolding** *ss*
unfolding *Opt.eqAct-def Van.eqAct-def*

apply(*simp-all add*: Δ*1-defs*)
by (*metis Nil-is-map-conv f3 infinity-ne-i0 llength-LNil ss3*)

show *match12-12* (*oor5* Δ*1* Δ*1′* Δ*2* Δ*3* Δ*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* Δ*1 sstat finals* **unfolding** *ss cfg statA′*
apply−**apply**(*frule* Δ*1-implies*)
apply(*simp add*: Δ*1-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*)
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* Δ*1* Δ*1′* Δ*2* Δ*3* Δ*e* ∞ ∞ ∞ *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* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-mispred*

then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
then show *?thesis* **using** *sa* Δ*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*: Δ*1-defs*)
subgoal by (*simp add*: Δ*1-defs*)
subgoal by (*simp add*: Δ*1-defs*)
subgoal using *xx-0-cases*[*of vs3*] **apply**(*elim disjE*)
subgoal by(*rule oor5I1*, *auto simp add*: Δ*1-defs*)
subgoal by(*rule oor5I1*, *auto simp add*: Δ*1-defs*) **.**
subgoal apply(*rule oor5I1*) **by** (*auto simp add*: Δ*1-defs*)
subgoal using *xx-NN-cases*[*of vs3*] **apply**(*elim disjE*)
subgoal by(*rule oor5I1*, *auto simp add*: Δ*1-defs*)
subgoal by(*rule oor5I1*, *auto simp add*: Δ*1-defs*) **.**
subgoal by(*rule oor5I1*, *auto simp add*: Δ*1-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
       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
       qed
      qed
     qed
    qed
qed
```

lemma *step2*: *unwindIntoCond* Δ*2* (*oor3* Δ*2* Δ*2'* Δ*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 Δ*2*: Δ*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*:¬*finalN ss1*
using Δ*2* **unfolding** *ss* Δ*2-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*2* **unfolding** *ss* Δ*2-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
using Δ*2* **unfolding** *ss*
apply−**apply**(*frule* Δ*2-implies*)
by (*simp add*: *finalS-if-spec*)

have *f4*:¬*finalS ss4*
using Δ*2* **unfolding** *ss*
apply−**apply**(*frule* Δ*2-implies*)
by (*simp add*: *finalS-if-spec*)

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*

then have *lpc3*:*pcOf* (*last cfgs3*) = *10* ∨
  pcOf (*last cfgs3*) = *3* ∨
  pcOf (*last cfgs3*) = *4* ∨
  pcOf (*last cfgs3*) = *11*
  using Δ*2* **unfolding** *ss* Δ*2-defs* **by** *simp*

have *sec3*[*simp*]:¬ *isSecO ss3*
  using Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
have *sec4*[*simp*]:¬ *isSecO ss4*
  using Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)

have *stat*[*simp*]:⋀*s3′ s4′ statA′*. *statA′* = *sstatA′ statA ss3 ss4* ⟹
  validTransO (*ss3*, *s3′*) ⟹ *validTransO* (*ss4*, *s4′*) ⟹
  (*statA* = *statA′* ∨ *statO* = *Diff*)
subgoal for *ss3′ ss4′*
  apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
  apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
  using Δ*2 finals* **unfolding** *ss* **apply** *clarsimp*
  apply(*simp-all add*: Δ*2-defs sstatA′-def*)
  apply(*cases statO*, *simp-all*) **by** (*cases statA*, *simp-all add*: *newStat-EqI*) **.**

have *xx*:*vs3 xx* = *vs4 xx* **using** Δ*2 lcfgs* **unfolding** *ss* Δ*2-defs* **apply** *clarsimp*
  by (*metis cfgs-Suc-zero config.sel*(*2*) *list.set-intros*(*1*) *state.sel*(*1*) *vstore.sel*)

have *oor3-rule*:⋀*ss3′ ss4′*. *ss3* →*S ss3′* ⟹ *ss4* →*S ss4′* ⟹
  (*pcOf* (*last cfgs3*) = *10* ⟶ *oor3* Δ*2* Δ*2′* Δ*1* ∞ *3 3 ss3′ ss4′*
(*sstatA′ statA ss3 ss4*) *ss1 ss2 statO*)
  ∧ (*pcOf* (*last cfgs3*) = *3* ∧ *mispred pstate4* [*6*, *3*] ⟶ *oor3* Δ*2* Δ*2′*
Δ*1* ∞ *2 2 ss3′ ss4′* (*sstatA′ statA ss3 ss4*) *ss1 ss2 statO*)
  ∧ (*pcOf* (*last cfgs3*) = *3* ∧ ¬*mispred pstate4* [*6*, *3*] ⟶ *oor3* Δ*2*
Δ*2′* Δ*1* ∞ *1 1 ss3′ ss4′* (*sstatA′ statA ss3 ss4*) *ss1 ss2 statO*)
  ∧ ((*pcOf* (*last cfgs3*) = *4* ∨ *pcOf* (*last cfgs3*) = *11*) ⟶ *oor3* Δ*2*
Δ*2′* Δ*1* ∞ *0 0 ss3′ ss4′* (*sstatA′ statA ss3 ss4*) *ss1 ss2 statO*) ⟹
  ∃ *w1′*<*w1*. ∃ *w2′*<*w2*. *oor3* Δ*2* Δ*2′* Δ*1* ∞ *w1′ w2′ 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 Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
          by (*metis enat-ord-simps*(*4*) *numeral-ne-infinity*)
        apply(*intro exI*[*of - 3*], *rule conjI*)

subgoal using Δ*2* **unfolding** *ss* Δ*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 Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis enat-ord-number*(*2*) *eval-nat-numeral*(*3*) *lessI*)
apply(*intro exI*[*of - 2*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis enat-ord-number*(*2*) *eval-nat-numeral*(*3*) *lessI*)
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify*
subgoal apply(*intro exI*[*of - 1*], *intro conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis one-less-numeral-iff semiring-norm*(*77*))
apply(*intro exI*[*of - 1*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis one-less-numeral-iff semiring-norm*(*77*))
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify* **.**
subgoal apply(*intro exI*[*of - 0*], *intro conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis less-numeral-extra*(*1*))
apply(*intro exI*[*of - 0*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis less-numeral-extra*(*1*))
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify* **. . . .**

show *react* (*oor3* Δ*2* Δ*2'* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor3* Δ*2* Δ*2'* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor3* Δ*2* Δ*2'* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor3* Δ*2* Δ*2'* Δ*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' cfgs3' ibT3' ibUT3' ls3'*
pstate4' cfg4' cfgs4' 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* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

```
next
  case spec-mispred
  then show ?thesis using stat Δ2 prem(6) unfolding ss by (auto simp add: Δ2-defs)
next
  case spec-Fence
  then show ?thesis using stat Δ2 prem(6) unfolding ss by (auto simp add: Δ2-defs)
next
  case spec-resolve
  then show ?thesis
    using Δ2 prem(6) resolve-106
    unfolding ss Δ2-defs apply clarify
    using cfgs-map misSpecL1-def
    by (smt (z3) insert-commute list.simps(15) resolve.simps)
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 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 sn3 Δ2 unfolding ss by (simp add: Δ2-defs, metis last-map)
  next
    case spec-mispred
    then show ?thesis using sn3 Δ2 unfolding ss by (simp add: Δ2-defs, metis last-map)
  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(frule 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
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 Δ2 unfolding ss by (auto simp add: Δ2-defs)

 next
  case spec-Fence
   then show ?thesis using stat Δ2 prem(6) unfolding ss by (auto simp
add: Δ2-defs)
 next
  case spec-normal
  then show ?thesis using stat Δ2 prem unfolding ss by (auto simp add:
Δ2-defs)
 next
  case spec-resolve
  then show ?thesis
   using Δ2 prem(6) resolve-63
   unfolding ss Δ2-defs using cfgs-map misSpecL1-def apply clarify
   by (smt (z3) insert-commute list.simps(15) resolve.simps)
 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 Δ2 unfolding ss by (simp add: Δ2-defs)
  next
   case nonspec-mispred
   then show ?thesis using sm3 Δ2 unfolding ss by (simp add: Δ2-defs)
  next
   case spec-resolve
   then show ?thesis using sm3 Δ2 unfolding ss by (simp add: Δ2-defs,
metis last-map)
  next
   case spec-Fence
    then show ?thesis using sm3 Δ2 unfolding ss apply−apply(frule
Δ2-implies)
   by (simp add: Δ2-defs)
  next
   case spec-normal
   then show ?thesis using sm3 Δ2 unfolding ss by (simp add: Δ2-defs,
metis last-map)
  next
   case spec-mispred note sm4 = spec-mispred
   have pc:pc4 = 3
    using prem(6) lcfgs Δ2 unfolding ss apply−apply(frule Δ2-implies)
```

```
      by (simp add: Δ2-defs )
    show ?thesis apply(rule oor3I2)
      unfolding ss Δ2'-def using xx-0-cases[of vs3] apply(elim disjE)
      subgoal using Δ2 lcfgs prem pc sm3 sm4 xx finals stat unfolding ss
        apply- apply(simp add: Δ2-defs Δ2'-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 Δ2 lcfgs prem pc sm3 sm4 xx finals stat unfolding ss
        apply- apply(simp add: Δ2-defs Δ2'-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
  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 Δ2 unfolding ss by (auto simp add: Δ2-defs)

   next
    case spec-Fence
      then show ?thesis using stat Δ2 prem(6) unfolding ss by (auto simp
add: Δ2-defs)
   next
    case spec-mispred
     then show ?thesis using stat Δ2 prem unfolding ss by (auto simp add:
Δ2-defs)
   next
    case spec-resolve
    then show ?thesis
      using Δ2 prem(6) resolve-63
      unfolding ss Δ2-defs using cfgs-map misSpecL1-def apply clarify
      by (smt (z3) insert-commute list.simps(15) resolve.simps)
```

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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-resolve*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-mispred*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-normal* **note** *sn4* = *spec-normal*
show *?thesis*
using Δ*2 lcfgs prem sn3 sn4 finals* **unfolding** *ss*
apply−**apply**(*frule* Δ*2-implies*) **apply** *clarify*
apply(*rule oor3I1*, *clarsimp*)
using *xx-0-cases*[*of vs3*] **apply**(*elim disjE*)
subgoal apply(*simp-all add*: Δ*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*: Δ*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* **. .**
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* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** Δ*2 prem* **unfolding** *ss* **by** *auto*
next
case *spec-normal*
then show *?thesis* **using** Δ*2 prem* **unfolding** *ss* **by** *auto*
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* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis* **using** *stat* Δ*2 prem sr3 sr4*
unfolding *ss lcfgs* **apply**−
apply(*frule* Δ*2-implies*) **apply** (*simp add*: Δ*2-defs* Δ*1-defs*)
apply(*rule oor3I3*, *simp add*: Δ*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 w1 w2 ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*3*: Δ*3 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*

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 Δ*3* **unfolding** *ss* Δ*3-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*3* **unfolding** *ss* Δ*3-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
using Δ*3* **unfolding** *ss*

apply−**apply**(*frule* Δ*3-implies*)
using *finalS-if-spec* **by** *force*

have *f4*:¬*finalS ss4*
using Δ*3* **unfolding** *ss*
apply−**apply**(*frule* Δ*3-implies*)
using *finalS-if-spec* **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*

then have *lpc3*:*pcOf* (*last cfgs3*) = *6* ∨
pcOf (*last cfgs3*) = *7*
using Δ*3* **unfolding** *ss* Δ*3-defs* **by** *simp*

have *sec3*[*simp*]:¬ *isSecO ss3*
using Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
have *sec4*[*simp*]:¬ *isSecO ss4*
using Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)

have *stat*[*simp*]:⋀*s3′ s4′ statA′*. *statA′* = *sstatA′ statA ss3 ss4* ⟹
validTransO (*ss3*, *s3′*) ⟹ *validTransO* (*ss4*, *s4′*) ⟹
(*statA* = *statA′* ∨ *statO* = *Diff*)
subgoal for *ss3′ ss4′*
apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
using Δ*3 finals* **unfolding** *ss* **apply** *clarsimp*
apply(*simp-all add*: Δ*3-defs sstatA′-def*)
apply(*cases statO*, *simp-all*) **by** (*cases statA*, *simp-all add*: *newStat-EqI*) **.**

have *vs3 xx* = *vs4 xx* **using** Δ*3 lcfgs* **unfolding** *ss* Δ*3-defs* **apply** *clarsimp*
by (*metis cfgs-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 Δ*3 lcfgs* **unfolding** *ss* Δ*3-defs array-loc-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 cfgs3*) = *6* ⟶ *oor* Δ*3* Δ*1* ∞ *1 1 ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*)
∧ (*pcOf* (*last cfgs3*) = *7* ⟶ *oor* Δ*3* Δ*1* ∞ *0 0 ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*)⟹
∃ *w1′*<*w1*. ∃ *w2′*<*w2*. *oor* Δ*3* Δ*1* ∞ *w1′ w2′ ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*

```
      subgoal for ss3' ss4' apply(cases ss3', cases ss4')
        subgoal for pstate3' cfg3' cfgs3' ib3' ls3'
              pstate4' cfg4' cfgs4' 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 . .

  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 unfolding ss by (auto simp add: Δ3-defs)

      next
        case nonspec-mispred
       then show ?thesis using stat Δ3 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-resolve
        then show ?thesis
```

using Δ*3 prem*(*6*) *resolve-106*
unfolding *ss* Δ*3-defs* **by** (*clarify*,*metis cfgs-map misSpecL1-def*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*3 prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
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* Δ*3 lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3 lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*3 lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*, *metis config.sel*(*1*) *last-map*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*3 lcfgs sn3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*, *metis config.sel*(*1*) *last-map*)
next
case *spec-resolve*
then show *?thesis* **using** *stat* Δ*3 lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-normal* **note** *sn4* = *spec-normal*
show *?thesis*
apply(*intro oorI1*)
unfolding *ss* Δ*3-def prem*(*4*,*5*) **apply** *clarify* **apply**− **apply**(*intro conjI*)
subgoal using *stat* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss prem*(*4*,*5*) *hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp-all add*: Δ*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* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss prem*(*4*,*5*) *hh*
apply− **apply**(*frule* Δ*3-implies*) **by**(*simp add*: Δ*3-defs*)
subgoal using *stat* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss hh*

apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*3-defs*)
using *cases-12*[*of pc3*] **apply** *simp* **apply**(*elim disjE*)
by *simp-all*
subgoal using *stat* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **by**(*simp add*: Δ*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** *stat* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*3 prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *stat* Δ*3 prem* **unfolding** *ss* Δ*3-defs* **apply** *simp*
by (*smt* (*verit,del-insts*) *cfgs-map empty-set insertCI insert-absorb list.set-map list.simps*(*15*) *numeral-eq-iff semiring-norm*(*87*,*89*) *set-ConsD singleton-insert-inj-eq′*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*3 prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
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* Δ*3 lcfgs sf3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3 lcfgs sf3* **unfolding** *ss* **by** (*simp add*:

```
Δ3-defs)
        next
          case spec-mispred
          then show ?thesis using stat Δ3 lcfgs sf3 unfolding ss
          apply (simp add: Δ3-defs)
          by (metis com.disc config.sel(1) last-map)
        next
          case spec-resolve
          then show ?thesis using stat Δ3 lcfgs sf3 unfolding ss
          by (simp add: Δ3-defs)
        next
          case spec-normal
          then show ?thesis using stat Δ3 lcfgs sf3 unfolding ss
          apply (simp add: Δ3-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 Δ1-def prem(4,5) apply− apply(clarify,intro conjI)
            subgoal using Δ3 lcfgs prem(1,2) sf3 sf4 unfolding ss hh
          apply− by(simp add: Δ3-defs Δ1-defs, metis ss stat validTransO.simps)

            subgoal using stat Δ3 lcfgs prem(4,5) sf3 sf4 unfolding ss hh
            apply− apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs)
            subgoal using stat Δ3 lcfgs prem(4,5) sf3 sf4 unfolding ss hh
            apply− apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs)
            subgoal using stat Δ3 lcfgs prem(4,5) sf3 sf4 unfolding ss hh
            apply− apply(frule Δ3-implies) by (simp add: Δ3-defs Δ1-defs) .
        qed

    qed . . .
  qed
qed
```

```
lemma step4: unwindIntoCond Δ1' Δ1
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 Δ1': Δ1' 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 Δ*1'* **unfolding** *ss* Δ*1'-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*1'* **unfolding** *ss* Δ*1'-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
  using Δ*1'* **unfolding** *ss*
  apply−**apply**(*frule* Δ*1'-implies*)
  by (*simp add*: *finalS-while-spec*)

have *f4*:¬*finalS ss4*
  using Δ*1'* **unfolding** *ss*
  apply−**apply**(*frule* Δ*1'-implies*)
  by (*simp add*: *finalS-while-spec*)

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 *match12-aux*:
(⋀*s1' s2' statA'*.

```
    statA' = sstatA' statA ss3 ss4 ⟹
    validTransO (ss3, s1') ⟹
    validTransO (ss4, s2') ⟹
    Opt.eqAct ss3 ss4 ⟹
    (¬ isSecO ss3 ∧ ¬ isSecO ss4 ∧
     (statA = statA' ∨ statO = Diff) ∧
      Δ1 ∞ 1 1 s1' s2' statA' ss1 ss2 statO))
  ⟹ match12 Δ1 w1 w2 ss3 ss4 statA ss1 ss2 statO
  apply(rule match12-simpleI, rule disjI1)

  apply(rule exI[of - 1], rule conjI)
    subgoal using Δ1' unfolding ss Δ1'-defs apply clarify
     by(metis enat-ord-simps(4) infinity-ne-i1)
  apply(rule exI[of - 1], rule conjI)
    subgoal using Δ1' unfolding ss Δ1'-defs apply clarify
      by(metis enat-ord-simps(4) infinity-ne-i1)
   by auto

 show react Δ1 w1 w2 ss3 ss4 statA ss1 ss2 statO
 unfolding react-def proof(intro conjI)

  show match1 Δ1 w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match1-def by (simp add: finalS-def final-def)
  show match2 Δ1 w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding match2-def by (simp add: finalS-def final-def)
  show match12 Δ1 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 ¬ isSecO ss3
    using v sa Δ1' unfolding ss by (simp add: Δ1'-defs, linarith)

    show ¬ isSecO ss4
```

using *v sa* $\Delta 1'$ **unfolding** *ss* **by** (*simp add*: $\Delta 1'$-*defs*, *linarith*)

show *stat*: *statA* = *statA'* ∨ *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*)

show $\Delta 1$ ∞ *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*(*2*) **unfolding** *ss*
apply (*simp add*: $\Delta 1'$-*defs*, *clarsimp*)
by (*smt* (*z3*) *insert-commute*)
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*, *metis*)
next
case *spec-normal*
then show *?thesis* **using** $\Delta 1'$ *sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 1'$*-defs*, *metis*)
next
case *spec-Fence*
then show *?thesis* **using** $\Delta 1'$ *sr3* **unfolding** *ss* **by** (*simp add*: $\Delta 1'$*-defs*, *metis*)
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 Δ*2′* **unfolding** *ss* Δ*2′-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*2′* **unfolding** *ss* Δ*2′-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
  using Δ*2′* **unfolding** *ss*
  apply−**apply**(*frule* Δ*2′-implies*)
  using *finalS-while-spec-L2* **by** *force*

have *f4*:¬*finalS ss4*
  using Δ*2′* **unfolding** *ss*
  apply−**apply**(*frule* Δ*2′-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 Δ*2′* **unfolding** *ss* **by** (*simp add*: Δ*2′-defs*)
have *sec4*[*simp*]:¬ *isSecO ss4*
    using Δ*2′* **unfolding** *ss* **by** (*simp add*: Δ*2′-defs*)

  have *stat*[*simp*]:⋀*s3′ s4′ statA′*. *statA′* = *sstatA′ statA ss3 ss4* ⟹
            validTransO (*ss3*, *s3′*) ⟹ *validTransO* (*ss4*, *s4′*) ⟹
            (*statA* = *statA′* ∨ *statO* = *Diff*)
  subgoal for *ss3′ ss4′*
    apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
      apply(*cases ss3′*, *cases ss4′*, *clarsimp*)
    using Δ*2′ finals* **unfolding** *ss* **apply** *clarsimp*

apply(*simp-all add*: Δ*2*′*-defs sstatA*′*-def*)
apply(*cases statO*, *simp-all*) **by** (*cases statA*, *simp-all add*: *newStat-EqI*) **.**

have *match12-aux*:
(⋀*pstate3*′ *cfg3*′ *cfgs3*′ *ib3*′ *ibUT3*′ *ls3*′
pstate4′ *cfg4*′ *cfgs4*′ *ib4*′ *ibUT4*′ *ls4*′ *statA*′.
(*pstate3*, *cfg3*, *cfgs3*, *ibT3*, *ibUT3*, *ls3*) →*S* (*pstate3*′, *cfg3*′, *cfgs3*′, *ib3*′, *ibUT3*′, *ls3*′) ⟹
(*pstate4*, *cfg4*, *cfgs4*, *ibT4*, *ibUT4*, *ls4*) →*S* (*pstate4*′, *cfg4*′, *cfgs4*′, *ib4*′, *ibUT4*′, *ls4*′) ⟹
Opt.eqAct ss3 ss4 ⟹ *statA*′ = *sstatA*′ *statA ss3 ss4* ⟹
(Δ*2* ∞ *1 1* (*pstate3*′, *cfg3*′, *cfgs3*′, *ib3*′, *ibUT3*′, *ls3*′) (*pstate4*′, *cfg4*′, *cfgs4*′, *ib4*′, *ibUT4*′, *ls4*′) *statA*′ *ss1 ss2 statO*))
⟹ *match12* Δ*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 Δ*2*′ **unfolding** *ss* Δ*2*′*-defs* **apply** *clarify*
by (*metis one-less-numeral-iff semiring-norm*(*76*))
apply(*rule exI*[*of - 1*], *rule conjI*)
subgoal using Δ*2*′ **unfolding** *ss* Δ*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* **. .**

show *react* Δ*2 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*2 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*2 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*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* Δ*2*′ **unfolding** *ss* **by** (*auto simp add*: Δ*2*′*-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2*′ **unfolding** *ss* **by** (*auto simp add*: Δ*2*′*-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*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*
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-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 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*)
apply *clarify* **apply**(*elim disjE*)
using *butlast.simps*(*2*) *insertCI last-ConsL last-map*
list.simps(*15*) *list.simps*(*8*) *map-L2 map-butlast not-Cons-self2*
resolve.simps resolve-106

```
          apply metis
          using butlast.simps(2) insertCI last-ConsL last-map
            list.simps(15) list.simps(8) map-L2 map-butlast not-Cons-self2
            resolve.simps resolve-106 apply metis
          using  butlast.simps(2) last.simps map-L2
            map-butlast map-is-Nil-conv neq-Nil-conv nth-Cons-0
            resolve-611 resolve-63 resolve-64
          by (metis last-map list.simps(15))
      qed
    qed .
  qed
qed


lemma stepe: unwindIntoCond Δe 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 Δe: Δe 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 ∧ finalN ss1 = finalS ss3 ∧ finalN ss2 = finalS ss4
    using Δe Opt.final-def finalS-def stepS-endPC endPC-def finalB-endPC
    unfolding Δe-defs ss by clarsimp
```

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

show *react* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
apply(*rule match12-simpleI*) **using** Δ*e* **unfolding** *ss* **apply** (*simp add*: Δ*e-defs*)
by (*simp add*: *stepS-endPC*)
qed
qed

lemmas *theConds* = *step0 step1 step2 step3 step4 step5 stepe*

proposition *lrsecure*
proof−
define *m* **where** *m*: *m* ≡ (*7*::*nat*)
define Δ*s* **where** Δ*s*: Δ*s* ≡ λ*i*::*nat*.
if i = *0 then* Δ*0*
else if i = *1 then* Δ*1*
else if i = *2 then* Δ*2*
else if i = *3 then* Δ*3*
else if i = *4 then* Δ*1'*
else if i = *5 then* Δ*2'*
else Δ*e*
define *nxt* **where** *nxt*: *nxt* ≡ λ*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* Δ*s*])
subgoal unfolding *m* **by** *auto*
subgoal unfolding *nxt m* **by** *auto*
subgoal using *init* **unfolding** Δ*s* **by** *auto*
subgoal
unfolding *m nxt* Δ*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* (‹⊥›)
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* ⇒ *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* ≡
[
0 Start ,
1 tt ::= (N 0),
2 xx ::= (N 1),
3 IfJump (Not (Eq (V xx) (N 0))) 4 13 ,

4/ *Input U xx* ,
5/ *Input T yy* ,
6/ *IfJump (Less (V xx) (N NN)) 7 12* ,
7/ *vv ::= VA aa1 (V xx)* ,
8/ *writeSecretOnFile*,
9/ *Fence* ,
10/ *tt ::= (VA aa2 (Times (V vv) (N 512)))* ,
11/ *Output U (V tt)* ,
12/ *Jump 3*,
13/ *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 ∧
pcOf cfg' = startOfWhileThen)

fun *rightWhileSpec* **where**
rightWhileSpec cfg cfg' =
(pcOf cfg = startOfWhileThen ∧
pcOf cfg' = whileElse)

fun *whileSpeculation* **where**
whileSpeculation cfg cfg' =
(leftWhileSpec cfg cfg' ∨
rightWhileSpec cfg cfg')
lemmas *whileSpec-def = whileSpeculation.simps*
startOfWhileThen-def
whileElse-def

lemmas *whileSpec-defs = whileSpec-def*
leftWhileSpec.simps
rightWhileSpec.simps

lemma *cases-14*: $(i{::}pcounter) = 0 \lor i = 1 \lor i = 2 \lor i = 3 \lor i = 4 \lor i = 5 \lor i = 6 \lor i = 7 \lor i = 8 \lor i = 9 \lor i = 10 \lor i = 11 \lor i = 12 \lor i = 13 \lor i = 14$

∨ *i* > *14*
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*)
subgoal for *i* **apply**(*cases i*, *simp-all*)
subgoal for *i* **apply**(*cases i*, *simp-all*)
. . . .

lemma *xx-0-cases*: *vs xx* = *0* ∨ *vs xx* ≠ *0* **by** *auto*

lemma *xx-NN-cases*: *vs xx* < *int NN* ∨ *vs xx* ≥ *int NN* **by** *auto*

lemma *is-If-pcOf*[*simp*]:
pcOf cfg < *14* ⟹ *is-IfJump* (*prog* ! (*pcOf cfg*)) ⟷ *pcOf cfg* = *3* ∨ *pcOf cfg* = *6*
apply(*cases cfg*) **using** *cases-14*[*of pcOf cfg*] **by** (*auto simp*: *prog-def*)

lemma *is-If-pc*[*simp*]:
pc < *14* ⟹ *is-IfJump* (*prog* ! *pc*) ⟷ *pc* = *3* ∨ *pc* = *6*
using *cases-14*[*of pc*] **by** (*auto simp*: *prog-def*)

lemma *eq-Fence-pc*[*simp*]:
pc < *14* ⟹ *prog* ! *pc* = *Fence* ⟷ *pc* = *9*
using *cases-14*[*of pc*] **by** (*auto simp*: *prog-def*)

lemma *output1*[*simp*]:*prog* ! *11* = *Output U* (*V tt*) **by**(*simp add*: *prog-def*)
lemma *output2*[*simp*]:*prog* ! *13* = *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*]:¬*is-IfJump* (*prog* ! *7*) **by**(*simp add*: *prog-def*)
lemma *is-nif2*[*simp*]:¬*is-IfJump* (*prog* ! *8*) **by**(*simp add*: *prog-def*)

lemma *getInput-not6*[*simp*]:¬ *is-getInput* (*prog* ! *6*) **by**(*simp add*: *prog-def*)
lemma *Output-not6*[*simp*]:¬ *is-Output* (*prog* ! *6*) **by**(*simp add*: *prog-def*)

lemma *getInput-not7*[*simp*]:¬ *is-getInput* (*prog* ! *7*) **by**(*simp add*: *prog-def*)
lemma *Output-not7*[*simp*]:¬ *is-Output* (*prog* ! *7*) **by**(*simp add*: *prog-def*)

lemma *getInput-not8*[*simp*]:¬ *is-getInput* (*prog* ! *8*) **by**(*simp add*: *prog-def*)

lemma *Output-not8*[*simp*]:*is-Output* (*prog* ! *8*) **by**(*simp add*: *prog-def*)

lemma *is-nif*[*simp*]:¬ *is-IfJump* (*prog* ! *9*) **by**(*simp add*: *prog-def*)
lemma *getInput-not10*[*simp*]:¬ *is-getInput* (*prog* ! *10*) **by**(*simp add*: *prog-def*)
lemma *Output-not10*[*simp*]:¬ *is-Output* (*prog* ! *10*) **by**(*simp add*: *prog-def*)

lemma *getInput-not12*[*simp*]:¬ *is-getInput* (*prog* ! *12*) **by**(*simp add*: *prog-def*)
lemma *Output-not12*[*simp*]:¬ *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* ≠ *Fence* **by**(*simp add*: *prog-def*)

consts *mispred* :: *predState* ⇒ *pcounter list* ⇒ *bool*
fun *resolve* :: *predState* ⇒ *pcounter list* ⇒ *bool* **where**
resolve p pc =
(*if* (*set pc* = {*4*,*13*} ∨ (*7* ∈ *set pc* ∧ (*4* ∈ *set pc* ∨ *13* ∈ *set pc*)) ∨ *pc* = [*12*,*8*])
then True else False)

lemma *resolve-73*: ¬*resolve p* [*7*,*3*] **by** *auto*
lemma *resolve-74*: *resolve p* [*7*,*4*] **by** *auto*
lemma *resolve-713*: *resolve p* [*7*,*13*] **by** *auto*
lemma *resolve-127*: ¬*resolve p* [*12*,*7*] **by** *auto*
lemma *resolve-129*: ¬*resolve p* [*12*,*9*] **by** *auto*

consts *update* :: *predState* ⇒ *pcounter list* ⇒ *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* = *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-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*,*cfgs*,*ibT*,*ibUT*,*ls*) ⟷
¬*finalS* (*pstate*,*cfg*,*cfgs*,*ibT*,*ibUT*,*ls*) ∧ *cfgs* = []
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* ⟹ *pc3* = *4* ⟹ *cfgs3* = [] ⟹ *cfgs4* = [] ⟹
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*)
⟹ *lhd ib3UT* = *lhd ib4UT*
using *getActO-pcOf zero-less-numeral* **by** *auto*

lemma *getActUntrustedInput*:*pc4* = *5* ⟹ *pc3* = *5* ⟹ *cfgs3* = [] ⟹ *cfgs4* = [] ⟹
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*)
⟹ *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 ≠ *0* ⟹
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* ≠ *0 then 13 else 4*) (*State* (*Vstore vs*) *avst hh p*), *ibT*, *ibUT*)
by(*cases vs xx* = *0*, *auto*)

lemma *nextB-pc4*[*simp*]:
ibUT ≠ *LNil* ⟹ *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 ≠ *LNil* ⟹ *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* ⟹
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 ≥ *int NN* ⟹
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* ⟹
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* ⟹
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* ⟹
pcOf (*last cfgs*) = *y* ⟹ *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* ⟹
pcOf (*cfgs* ! *0*) = *x* ⟹
pcOf (*last cfgs*) = *y* ⟹ *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 *length cfgs* = *2* ⟹ (*cfgs* ! *0*) = *last* (*butlast cfgs*)
by (*cases cfgs*, *auto*)

lemma *nextB-stepB-pc*:
pc < *14* ⟹ (*pc* = *4* ⟶ *ibUT* ≠ *LNil*) ⟹ (*pc* = *5* ⟶ *ibT* ≠ *LNil*) ⟹
(*Config pc s*, *ibT*, *ibUT*) →*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*)

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(*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(*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(*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*
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 *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 *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-all* **. .**

lemma *not-finalB*:
$pc < 14 \Longrightarrow (pc = 4 \longrightarrow ibUT \neq LNil) \Longrightarrow (pc = 5 \longrightarrow ibT \neq LNil) \Longrightarrow$
$\neg$ *finalB* (*Config pc s*, *ibT*, *ibUT*)
using *nextB-stepB-pc* **by** (*simp add*: *stepB-iff-nextB*)

lemma *finalB-pc-iff'*:
pc < *14* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *4* ∧ *ibUT* = *LNil*) ∨ (*pc* = *5* ∧ *ibT* = *LNil*)
apply *standard*
subgoal using *nextB-stepB-pc*[*of pc*] **by** (*auto simp add*: *stepB-iff-nextB*)
unfolding *finalB-iff* **by** (*elim disjE*, *simp-all add*: *prog-def*)

lemma *finalB-pc-iff*:
pc ≤ *14* ⟹
finalB (*Config pc s*, *ibT*, *ibUT*) ⟷
(*pc* = *14* ∨ (*pc* = *4* ∧ *ibUT* = *LNil*) ∨ (*pc* = *5* ∧ *ibT* = *LNil*))
using *Prog.finalB-iff endPC finalB-pc-iff' order-le-less finalB-iff*
by *metis*

lemma *finalB-pcOf-iff*[*simp*]:
pcOf cfg ≤ *14* ⟹
finalB (*cfg*, *ibT*, *ibUT*) ⟷ (*pcOf cfg* = *14* ∨ (*pcOf cfg* = *4* ∧ *ibUT* = *LNil*) ∨ (*pcOf cfg* = *5* ∧ *ibT* = *LNil*))
using *config.collapse finalB-pc-iff* **by** *metis*

lemma *finalS-cond*:*pcOf cfg* < *14* ⟹ *noMisSpec cfgs* ⟹ *ibT* ≠ *LNil* ⟹ *ibUT* ≠ *LNil* ⟹ ¬ *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* ∪ *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* ∪ *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* ∪ *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*)
pstate update pstate [*pcOf* (*Config pc* (*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 Config (*if vs xx* ≠ *0 then 13 else 4*) (*State* (*Vstore vs*) *avst hh p*)
ibT ibUT [*Config* (*if vs xx* ≠ *0 then 13 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 cfgs Config pc* (*State* (*Vstore vs*) *avst hh p*)
pstate pstate ibT ibUT
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*)) *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 5 (*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 cfgs Config pc* (*State* (*Vstore vs*) *avst hh p*)
pstate pstate ibT ibUT
Config 6 (*State* (*Vstore* (*vs*(*yy*:= *lhd ibT*))) *avst hh p*)
ltl 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* [*6*])

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 xx* < *NN then 7 else 12*) (*State* (*Vstore vs*) *avst hh p*)
ibT ibUT Config (*if vs xx* < *NN then 12 else 7*) (*State* (*Vstore vs*) *avst hh p*)
ibT ibUT [*Config* (*if vs xx* < *NN then 12 else 7*) (*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 xx* < *NN then 7 else 12*) (*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 xx*)) (*Avstore as*))
in (*Config 8* (*State* (*Vstore* (*vs*(*vv* := *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 9* (*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 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* ∪ *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* ⟹
¬ *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 using *stepS-spec-resolve-iff*[*of cfgs pstate cfg ibT ibUT ls update pstate* (*pcOf cfg* # *map pcOf cfgs*)]
by (*metis* (*no-types*, *lifting*) *cfgs-map empty-set insert-commute less-numeral-extra*(*3*)

resolve.simps list.simps(*15*) *list.size*(*3*) *numeral-2-eq-2 pos2*)

subgoal apply(*elim conjE*)
using *spec-resolve*[*of cfgs pstate cfg update pstate* (*pcOf cfg* # *map pcOf cfgs*) *cfg* [] *ibT ibT ibUT ibUT ls ls*]
using *empty-set resolve.simps length-0-conv*
length-1-butlast length-Suc-conv list.simps(*9*,*15*)
cfgs-map not-Cons-self2 spec-resolve **by** *metis* **. . . .**

lemma *finalS-while-spec-L2*:
pcOf cfg = *7* ⟹
whileSpeculation (*cfgs*!*0*) (*last cfgs*) ⟹
length cfgs = *2* ⟹
¬ *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 using *stepS-spec-resolve-iff*[*of cfgs pstate cfg ibT ibUT ls update pstate* (*pcOf cfg* # *map pcOf cfgs*)]
unfolding *resolve.simps*
using *list.set-intros*(*1*,*2*) *map-L2 zero-neq-numeral*
by *fastforce*
subgoal apply(*elim conjE*)
using *spec-resolve*
unfolding *resolve.simps*
using *list.set-intros*(*1*,*2*) *map-L2 zero-neq-numeral*
by (*metis* (*no-types*, *lifting*) *Prog-Mispred.spec-resolve Prog-Mispred-axioms list.size*(*3*))
. . . .

lemma *finalS-if-spec*:
(*pcOf* (*last cfgs*) ∈ *inThenIfBeforeOutput* ∧ *pcOf cfg* = *12*) ∨
(*pcOf* (*last cfgs*) ∈ *inElseIf* ∧ *pcOf cfg* = *7*) ⟹
length cfgs = *Suc 0* ⟹
¬ *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(*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* ∪ *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* ∪ *readLocs* (*last cfgs*))])
by(*erule notE*, *rule spec-resolve*, *auto*) **.**
subgoal apply(*elim conjE*, *elim disjE*)
subgoal apply(*rule notI*, *erule allE*[*of* -
(*pstate*, *cfg*, [*Config 3* (*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 3* (*State* (*Vstore vs*) *avst hh p*)], *auto*)

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*)

subgoal by (*metis resolve-74 stepS-spec-resolve-iff*
map-L1 cfgs-Suc-zero not-Cons-self2)
subgoal by (*metis resolve-713 stepS-spec-resolve-iff*
map-L1 cfgs-Suc-zero not-Cons-self2)
. . . . .

end

13.2 Proof

theory *Fun6-secure*
imports *Fun6*
begin

definition *common* :: *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool*
where
common = (λ*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* ∧
cfg1 = *cfg3* ∧ *cfg2* = *cfg4* ∧
pcOf cfg3 = *pcOf cfg4* ∧ *map pcOf cfgs3* = *map pcOf cfgs4* ∧
pcOf cfg3 ∈ *PC* ∧ *pcOf* ' (*set cfgs3*) ⊆ *PC* ∧
llength ibT1 = ∞ ∧ *llength ibT2* = ∞ ∧
llength ibUT1 = ∞ ∧ *llength ibUT2* = ∞ ∧

ibT1 = *ibT3* ∧ *ibT2* = *ibT4* ∧
ibUT1 = *ibUT3* ∧ *ibUT2* = *ibUT4* ∧

w1 = *w2* ∧
///
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*))) ∧
(∀ *cfg4'*∈*set cfgs4*. *array-base aa2* (*getAvstore* (*stateOf cfg4'*)) = *array-base aa2* (*getAvstore* (*stateOf cfg4*))) ∧
///
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
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 ⟹
pcOf cfg1 < *14* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
ibT1 ≠ [[]] ∧ *ibT2* ≠ [[]] ∧
ibUT1 ≠ [[]] ∧ *ibUT2* ≠ [[]] ∧

w1 = *w2*
unfolding *common-def PC-def* **by** (*auto simp*: *image-def subset-eq*)

definition Δ*0* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*0* = (λ*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 ∧
 pcOf cfg3 ∈ *beforeWhile* ∧
 (*pcOf cfg3* > *1* ⟶ *same-var-o tt cfg3 cfgs3 cfg4 cfgs4*) ∧
 (*pcOf cfg3* > *2* ⟶ *same-var-o xx cfg3 cfgs3 cfg4 cfgs4*) ∧
 (*pcOf cfg3* > *4* ⟶ *same-var-o xx cfg3 cfgs3 cfg4 cfgs4*) ∧
 noMisSpec cfgs3
))

lemmas Δ*0-defs* = Δ*0-def common-def PC-def same-var-o-def*
 beforeWhile-def noMisSpec-def

lemma Δ*0-implies*: Δ*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 ⟹
 pcOf cfg1 < *14* ∧ *pcOf cfg2* = *pcOf cfg1* ∧
 ibT1 ≠ [[]] ∧ *ibT2* ≠ [[]] ∧
 ibUT1 ≠ [[]] ∧ *ibUT2* ≠ [[]] ∧
 cfgs4 = []
 apply (*meson* Δ*0-def common-implies*)
 by (*simp-all add*: Δ*0-defs*, *metis Nil-is-map-conv*)

definition Δ*1* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*1* = (λ*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 ∧
pcOf cfg3 ∈ *afterWhile* ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
noMisSpec cfgs3
))
lemmas Δ*1-defs* = Δ*1-def common-def noMisSpec-def PC-def afterWhile-def same-var-o-def*
lemma Δ*1-implies*: Δ*1 n w1 w2* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf cfg3 < *14* ∧ *cfgs3* = [] ∧ *ibT3* ≠ [[]] ∧
pcOf cfg4 < *14* ∧ *cfgs4* = [] ∧ *ibT4* ≠ [[]] ∧
ibUT3 ≠ [[]] ∧ *ibUT4* ≠ [[]]
unfolding Δ*1-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 Δ*1′* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV*
⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*1′* = (λ*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 ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
whileSpeculation cfg3 (*last cfgs3*) ∧
misSpecL1 cfgs3 ∧ *misSpecL1 cfgs4* ∧
w1 = ∞
))

lemmas Δ*1′-defs* = Δ*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 $\Longrightarrow$
pcOf cfg3 < *14* ∧ *pcOf cfg4* < *14* ∧
whileSpeculation cfg3 (*last cfgs3*) ∧
whileSpeculation cfg4 (*last cfgs4*) ∧
length cfgs3 = *Suc 0* ∧ *length cfgs4* = *Suc 0*
unfolding $\Delta 1'$*-defs* **by** *clarsimp*

definition $\Delta 2$ :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
$\Delta 2$ = (λ*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 ∧

same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfIfThen* ∧ *pcOf* (*last cfgs3*) ∈ *inElseIf* ∧
misSpecL1 cfgs3 ∧ *misSpecL1 cfgs4* ∧

(*pcOf* (*last cfgs3*) = *startOfElseBranch* ⟶ *w1* = ∞) ∧
(*pcOf* (*last cfgs3*) = *3* ⟶ *w1* = *3*) ∧

(*pcOf* (*last cfgs3*) = *startOfWhileThen* ∨
pcOf (*last cfgs3*) = *whileElse* ⟶ *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 ⟹
pcOf (*last cfgs3*) ∈ *inElseIf* ∧ *pcOf cfg3* = *7* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg4 = *pcOf cfg3* ∧ *length cfgs3* = *Suc 0* ∧
length cfgs4 = *Suc 0* ∧ *same-var xx* (*last cfgs3*) (*last cfgs4*)
apply(*intro conjI*)
unfolding Δ*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 Δ*2′* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*2′* = (λ*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 ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfIfThen* ∧
whileSpeculation (*cfgs3*!*0*) (*last cfgs3*) ∧
misSpecL2 cfgs3 ∧ *misSpecL2 cfgs4* ∧
w1 = *2*
))

lemmas Δ*2′-defs* = Δ*2′-def common-def PC-def same-var-def*
startOfElseBranch-def startOfIfThen-def
whileSpec-defs misSpecL2-def

lemma Δ*2′-implies*: Δ*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 ⟹
pcOf cfg3 = *7* ∧ *pcOf cfg4* = *7* ∧
whileSpeculation (*cfgs3*!*0*) (*last cfgs3*) ∧
whileSpeculation (*cfgs4*!*0*) (*last cfgs4*) ∧

length cfgs3 = *2* ∧ *length cfgs4* = *2*
apply(*intro conjI*)
unfolding Δ*2'-defs* **apply** (*simp add*: *lessI*, *clarify*)
apply *linarith*+ **apply** *simp-all*
by (*metis list.inject map-L2*)

definition Δ*3* :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δ*3* = (λ*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 ∧
same-var-o xx cfg3 cfgs3 cfg4 cfgs4 ∧
pcOf cfg3 = *startOfElseBranch* ∧ *pcOf* (*last cfgs3*) ∈ *inThenIfBeforeOutput* ∧
misSpecL1 cfgs3 ∧
(*pcOf* (*last cfgs3*) = *7* ⟶ *w1* = ∞) ∧
(*pcOf* (*last cfgs3*) = *8* ⟶ *w1* = *2*) ∧
(*pcOf* (*last cfgs3*) = *9* ⟶ *w1* = *1*)
))

lemmas Δ*3-defs* = Δ*3-def common-def PC-def same-var-o-def*
startOfElseBranch-def inThenIfBeforeOutput-def

lemma Δ*3-implies*: Δ*3 num w1 w2* (*pstate3*,*cfg3*,*cfgs3*,*ibT3*,*ibUT3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ibT4*,*ibUT4*,*ls4*)
statA
(*cfg1*,*ibT1*,*ibUT1*,*ls1*)
(*cfg2*,*ibT2*,*ibUT2*,*ls2*)
statO ⟹
pcOf (*last cfgs3*) ∈ *inThenIfBeforeOutput* ∧
pcOf (*last cfgs4*) = *pcOf* (*last cfgs3*) ∧
pcOf cfg3 = *12* ∧ *pcOf cfg3* = *pcOf cfg4* ∧
length cfgs3 = *Suc 0* ∧ *length cfgs4* = *Suc 0*
apply(*intro conjI*)
unfolding Δ*3-defs*
apply (*simp-all add*: *image-subset-iff*)
by (*metis last-map map-is-Nil-conv length-map*)+

definition Δe :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *stateO* ⇒ *stateO* ⇒ *status* ⇒ *stateV* ⇒ *stateV* ⇒ *status* ⇒ *bool* **where**
Δe = (λ*num w1 w2* (*pstate3*,*cfg3*,*cfgs3*,*ib3*,*ls3*)
(*pstate4*,*cfg4*,*cfgs4*,*ib4*,*ls4*)
statA
(*cfg1*,*ib1*,*ls1*)
(*cfg2*,*ib2*,*ls2*)
statO.
(*pcOf cfg3* = *endPC* ∧ *pcOf cfg4* = *endPC* ∧ *cfgs3* = [] ∧ *cfgs4* = [] ∧
pcOf cfg1 = *endPC* ∧ *pcOf cfg2* = *endPC*))

lemmas Δ*e-defs* = Δ*e-def common-def endPC-def*

lemma *init*: *initCond* Δ*0*
unfolding *initCond-def* **apply** *safe*
subgoal for *pstate3 cfg3 cfgs3 ibT3 ibUT3 ls3*
pstate4 cfg4 cfgs4 ibT4 ibUT4 ls4
unfolding *istateO.simps* **apply** *clarsimp*
apply(*cases getAvstore* (*stateOf cfg3*), *cases getAvstore* (*stateOf cfg4*))
unfolding Δ*0-defs*
unfolding *array-base-def* **by** *auto* **.**

lemma *step0*: *unwindIntoCond* Δ*0* (*oor* Δ*0* Δ*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 Δ*0*: Δ*0 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 Δ*0* **unfolding** *ss*
  apply−**by**(*frule* Δ*0-implies*, *simp*)

have *f2*:¬*finalN ss2*
  using Δ*0* **unfolding** *ss*
  apply−**by**(*frule* Δ*0-implies*, *simp*)

have *f3*:¬*finalS ss3*
  using Δ*0* **unfolding** *ss*
  apply−**apply**(*frule* Δ*0-implies*, *unfold* Δ*0-defs*)
  using *finalS-cond′* **by** *simp*

have *f4*:¬*finalS ss4*
  using Δ*0* **unfolding** *ss*
  apply−**apply**(*frule* Δ*0-implies*, *unfold* Δ*0-defs*)
  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* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

  show *match1* (*oor* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
  unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
  show *match2* (*oor* Δ*0* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
  unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
  show *match12* (*oor* Δ*0* Δ*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* = *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* Δ*0 finals* **unfolding** *ss*
by (*simp add*: Δ*0-defs eqSec-def*)

show *eqSec ss2 ss4*
using *v sa* Δ*0 finals* **unfolding** *ss*
by (*simp add*: Δ*0-defs eqSec-def*, *metis map-is-Nil-conv*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*0* **unfolding** *ss*
apply−**apply**(*frule* Δ*0-implies*)
unfolding *Opt.eqAct-def*
Van.eqAct-def
by(*simp-all add*: Δ*0-defs*, *linarith*)

show *match12-12* (*oor* Δ*0* Δ*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* Δ*0 sstat* **unfolding** *ss cfg statA'* **apply** *simp*
apply(*simp add*: Δ*0-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* Δ*0* Δ*1* ∞ ∞ ∞ *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* Δ*0 stat* **unfolding** *ss*
by (*simp add*: Δ*0-defs numeral-1-eq-Suc-0*, *linarith*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*0 stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*0-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
show *?thesis* **using** *sa* Δ*0 stat v3 v4 nn3 nn4* **unfolding** *ss cfg* **apply** *clarsimp*
apply(*unfold* Δ*0-defs*, *clarsimp*, *elim disjE*)

```
          subgoal by(rule oorI1, auto simp add: Δ0-defs)
          subgoal by (rule oorI1, simp add: Δ0-defs)
          subgoal by (rule oorI2, simp add: Δ1-defs) .
        qed
      qed
    qed
  qed
 qed
qed
```

```
lemma step1: unwindIntoCond Δ1 (oor5 Δ1 Δ1' Δ2 Δ3 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 Δ1: Δ1 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 Δ1 unfolding ss Δ1-def apply clarify
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by metis

  have f2:¬finalN ss2
  using Δ1 unfolding ss Δ1-def apply clarify
```

```
  apply(frule common-implies)
  using finalB-pcOf-iff finalN-iff-finalB nat-less-le by metis


  have f3:¬finalS ss3
    using Δ1 unfolding ss
    apply−apply(frule Δ1-implies)
    by (simp add: finalS-cond′)

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

  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 (oor5 Δ1 Δ1′ Δ2 Δ3 Δe) w1 w2 ss3 ss4 statA ss1 ss2 statO
  unfolding react-def proof(intro conjI)

    show match1 (oor5 Δ1 Δ1′ Δ2 Δ3 Δe) w1 w2 ss3 ss4 statA ss1 ss2 statO
    unfolding match1-def by (simp add: finalS-def final-def)
    show match2 (oor5 Δ1 Δ1′ Δ2 Δ3 Δe) w1 w2 ss3 ss4 statA ss1 ss2 statO
    unfolding match2-def by (simp add: finalS-def final-def)
    show match12 (oor5 Δ1 Δ1′ Δ2 Δ3 Δe) 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′

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

      show eqSec ss2 ss4
```

using *v sa* Δ*1 finals* **unfolding** *ss*
by (*simp add*: Δ*1-defs eqSec-def*, *metis map-is-Nil-conv*)

show *Van.eqAct ss1 ss2*
using *v sa* Δ*1* **unfolding** *ss* **apply**− **apply**(*frule* Δ*1-implies*)
unfolding *Opt.eqAct-def Van.eqAct-def*
apply(*simp-all add*: Δ*1-defs*)
by (*metis f3 getActO-pcOf numeral-eq-iff numeral-less-iff semiring-norm*(*77*,*78*,*81*,*89*) *ss3*)

show *match12-12* (*oor5* Δ*1* Δ*1′* Δ*2* Δ*3* Δ*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* Δ*1 sstat finals* **unfolding** *ss cfg statA′*
apply−**apply**(*frule* Δ*1-implies*)
apply(*simp add*: Δ*1-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* Δ*1* Δ*1′* Δ*2* Δ*3* Δ*e* ∞ ∞ ∞ *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* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1* *stat* **unfolding** *ss* **by** (*simp add*: Δ*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* Δ*1* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sa* Δ*1* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *sa* Δ*1* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sa* Δ*1* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *spec-resolve*
then show *?thesis* **using** *sa* Δ*1* *stat nn3* **unfolding** *ss* **by** (*simp add*: Δ*1-defs*)
next
case *nonspec-normal* **note** *nn4* = *nonspec-normal*
then show *?thesis* **using** *sa* Δ*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: Δ1-defs)
        subgoal by (simp add: Δ1-defs)
        subgoal by (simp add: Δ1-defs)
        subgoal using xx-0-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 apply(rule oor5I1) by (auto simp add: Δ1-defs)
        subgoal apply(rule oor5I1) by (auto simp add: Δ1-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 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
  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-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: Δ1-defs Δ1'-defs)
```

```
          subgoal by(rule oor5I2, auto simp add: Δ1-defs Δ1'-defs) .
        prefer 6 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
     qed
    qed
   qed
  qed
qed
```

```
lemma step2: unwindIntoCond Δ2 (oor3 Δ2 Δ2' Δ1)
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 Δ2: Δ2 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:¬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 Δ*2* **unfolding** *ss*
  apply−**apply**(*frule* Δ*2-implies*)
  by (*simp add*: *finalS-if-spec*)

have *f4*:¬*finalS ss4*
  using Δ*2* **unfolding** *ss*
  apply−**apply**(*frule* Δ*2-implies*)
  by (*simp add*: *finalS-if-spec*)

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*

then have *lpc3*:*pcOf* (*last cfgs3*) = *12* ∨
      pcOf (*last cfgs3*) = *3* ∨
      pcOf (*last cfgs3*) = *4* ∨
      pcOf (*last cfgs3*) = *13*
  using Δ*2* **unfolding** *ss* Δ*2-defs* **by** *simp*

have *sec3*[*simp*]:¬ *isSecO ss3*
  using Δ*2* *finals* **unfolding** *ss isSecO-def*
  by(*simp add*: Δ*2-defs*, *metis list.size*(*3*) *n-not-Suc-n*)

have *sec4*[*simp*]:¬ *isSecO ss4*
  using Δ*2* **unfolding** *ss*
  by (*simp add*: Δ*2-defs*, *metis list.size*(*3*) *n-not-Suc-n*)

  have *stat*[*simp*]:⋀*s3′ s4′ statA′*. *statA′* = *sstatA′ statA ss3 ss4* ⟹
      validTransO (*ss3*, *s3′*) ⟹ *validTransO* (*ss4*, *s4′*) ⟹
      (*statA* = *statA′* ∨ *statO* = *Diff*)
  subgoal for *ss3′ ss4′*
    apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
    apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
    using Δ*2* *finals* **unfolding** *ss* **apply** *clarsimp*
    apply(*simp-all add*: Δ*2-defs sstatA′-def*)
    apply(*cases statO*, *simp-all*) **by** (*cases statA*, *simp-all add*: *newStat-EqI*) **.**

have *xx*:*vs3 xx* = *vs4 xx* **using** Δ*2* *lcfgs* **unfolding** *ss* Δ*2-defs* **apply** *clarsimp*
  by (*metis cfgs-Suc-zero config.sel*(*2*) *list.set-intros*(*1*) *state.sel*(*1*) *vstore.sel*)

have *oor3-rule*:⋀*ss3′ ss4′*. *ss3* →*S ss3′* ⟹ *ss4* →*S ss4′* ⟹
        (*pcOf* (*last cfgs3*) = *12* ⟶ *oor3* Δ*2* Δ*2′* Δ*1* ∞ *3 3 ss3′ ss4′*

(*sstatA' statA ss3 ss4*) *ss1 ss2 statO*)
∧ (*pcOf* (*last cfgs3*) = *3* ∧ *mispred pstate4* [*7*, *3*] ⟶ *oor3* Δ*2* Δ*2'* Δ*1* ∞ *2 2 ss3' ss4'* (*sstatA' statA ss3 ss4*) *ss1 ss2 statO*)
∧ (*pcOf* (*last cfgs3*) = *3* ∧ ¬*mispred pstate4* [*7*, *3*] ⟶ *oor3* Δ*2* Δ*2'* Δ*1* ∞ *1 1 ss3' ss4'* (*sstatA' statA ss3 ss4*) *ss1 ss2 statO*)
∧ ((*pcOf* (*last cfgs3*) = *4* ∨ *pcOf* (*last cfgs3*) = *13*) ⟶ *oor3* Δ*2* Δ*2'* Δ*1* ∞ *0 0 ss3' ss4'* (*sstatA' statA ss3 ss4*) *ss1 ss2 statO*) ⟹
∃ *w1'*<*w1*. ∃ *w2'*<*w2*. *oor3* Δ*2* Δ*2'* Δ*1* ∞ *w1' w2' 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 Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
by (*metis enat-ord-simps*(*4*) *numeral-ne-infinity*)
apply(*intro exI*[*of - 3*], *rule conjI*)
subgoal using Δ*2* **unfolding** *ss* Δ*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 Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis enat-ord-number*(*2*) *eval-nat-numeral*(*3*) *lessI*)
apply(*intro exI*[*of - 2*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis enat-ord-number*(*2*) *eval-nat-numeral*(*3*) *lessI*)
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify*
subgoal apply(*intro exI*[*of - 1*], *intro conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis one-less-numeral-iff semiring-norm*(*77*))
apply(*intro exI*[*of - 1*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis one-less-numeral-iff semiring-norm*(*77*))
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify* **.**
subgoal apply(*intro exI*[*of - 0*], *intro conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis less-numeral-extra*(*1*))
apply(*intro exI*[*of - 0*], *rule conjI*)
using Δ*2* **unfolding** *ss* Δ*2-defs* **apply** *clarify*
apply (*metis less-numeral-extra*(*1*))
using Δ*2 p* **unfolding** *ss* Δ*2-defs* **by** *clarify* **. . . .**

show *react* (*oor3* Δ*2* Δ*2'* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor3* Δ*2* Δ*2'* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*

unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor3* Δ*2* Δ*2′* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor3* Δ*2* Δ*2′* Δ*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′ cfgs3′ ibT3′ ibUT3′ ls3′*
pstate4′ cfg4′ cfgs4′ 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* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2* *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2* *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-resolve*
then show *?thesis*
using Δ*2* *prem*(*6*) **unfolding** *ss* **apply** (*simp add*: Δ*2-defs*, *clarsimp*)
by (*meson doubleton-eq-iff numeral-eq-iff semiring-norm*(*89*) *semiring-norm*(*90*))
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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-resolve*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)

next
case *spec-mispred*
then show *?thesis* **using** *sn3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-normal* **note** *sn4* = *spec-normal*
have *pc4*:*pc4* = *12* **using** Δ*2 prem lcfgs* **unfolding** *ss* Δ*2-defs* **by** *auto*
show *?thesis*
using Δ*2 prem sn3 sn4 finals stat* **unfolding** *ss prem*(*4*,*5*) *lcfgs*
apply−**apply**(*frule* Δ*2-implies*, *unfold* Δ*2-defs*) **apply** *clarsimp*
apply(*rule oor3I1*) **apply**(*simp-all add*: Δ*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* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2 prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*2 prem* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis cfgs-map*)
next
case *spec-resolve*
then show *?thesis*
using Δ*2 prem*(*6*) *resolve-73*
unfolding *ss* Δ*2-defs* **using** *cfgs-map misSpecL1-def*
by (*clarify*,*smt* (*z3*) *insert-commute list.simps*(*15*) *resolve.simps*)
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* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *sm3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)

next
case *spec-resolve*
then show *?thesis* **using** *sm3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-Fence*
then show *?thesis* **using** *sm3* Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-implies*)
by (*simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *sm3* Δ*2* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis last-map*)
next
case *spec-mispred* **note** *sm4* = *spec-mispred*
have *pc*:*pc4* = *3*
using *prem*(*6*) *lcfgs* Δ*2* **unfolding** *ss* **apply**−**apply**(*frule* Δ*2-implies*)
by (*simp add*: Δ*2-defs*)
show *?thesis* **apply**(*rule oor3I2*)
unfolding *ss* Δ*2′-def* **using** *xx-0-cases*[*of vs3*] **apply**(*elim disjE*)
subgoal using Δ*2 lcfgs prem pc sm3 sm4 xx finals stat* **unfolding** *ss*
apply− **apply**(*simp add*: Δ*2-defs* Δ*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 Δ*2 lcfgs prem pc sm3 sm4 xx finals stat* **unfolding** *ss*
apply− **apply**(*simp add*: Δ*2-defs* Δ*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* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*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*
then show *?thesis* **using** *stat* $\Delta 2$ *prem* **unfolding** *ss* **by** (*auto simp add*: $\Delta 2$*-defs*)
next
case *spec-resolve*
then show *?thesis*
using $\Delta 2$ *prem*(*6*) *resolve-73*
unfolding *ss* $\Delta 2$*-defs* **using** *cfgs-map misSpecL1-def*
by (*clarify*,*smt* (*z3*) *insert-commute list.simps*(*15*) *resolve.simps*)
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* $\Delta 2$ **unfolding** *ss* **by** (*simp add*: $\Delta 2$*-defs*, *metis last-map*)
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*, *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*) *cfgs-Suc-zero lcfgs list.set-intros*(*1*))
apply (*smt* (*verit*) *cfgs-Suc-zero lcfgs list.set-intros*(*1*))
apply *presburger*
apply (*smt* (*verit*) *insertCI list.simps*(*15*) *resolve.elims*(*3*) *resolve-74 resolve-127*)
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* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)

next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*2 prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2-defs*)
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* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2 sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*)
next
case *spec-normal*

then show *?thesis* **using** *stat* Δ*2* *sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2* *sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2* *sr3* **unfolding** *ss* **by** (*simp add*: Δ*2-defs*, *metis*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis* **using** *stat* Δ*2* *prem sr3 sr4*
unfolding *ss lcfgs* **apply**−
apply(*frule* Δ*2-implies*) **apply** (*simp add*: Δ*2-defs* Δ*1-defs*)
apply(*rule oor3I3*, *simp add*: Δ*1-defs*)
by (*metis 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 w1 w2 ss3 ss4 statA ss1 ss2 statO*
assume *r*: *reachO ss3 reachO ss4 reachV ss1 reachV ss2*
and Δ*3*: Δ*3 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*

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 Δ*3* **unfolding** *ss* Δ*3-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*3* **unfolding** *ss* Δ*3-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
  using Δ*3* **unfolding** *ss*
  apply−**apply**(*frule* Δ*3-implies*)
  using *finalS-if-spec* **by** *force*

have *f4*:¬*finalS ss4*
  using Δ*3* **unfolding** *ss*
  apply−**apply**(*frule* Δ*3-implies*)
  using *finalS-if-spec* **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*

then have *lpc3*:*pcOf* (*last cfgs3*) = *7* ∨
        pcOf (*last cfgs3*) = *8*
  using Δ*3* **unfolding** *ss* Δ*3-defs* **by** *simp*

have *sec3*[*simp*]:¬ *isSecO ss3*
 using Δ*3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*, *metis list.size*(*3*) *n-not-Suc-n*)

have *sec4*[*simp*]:¬ *isSecO ss4*
  using Δ*3* **unfolding** *ss*
  by (*simp add*: Δ*3-defs*, *metis list.size*(*3*) *map-is-Nil-conv nat.distinct*(*1*))

  have *stat*[*simp*]:⋀*s3' s4' statA'*. *statA'* = *sstatA' statA ss3 ss4* ⟹
      validTransO (*ss3*, *s3'*) ⟹ *validTransO* (*ss4*, *s4'*) ⟹
      (*statA* = *statA'* ∨ *statO* = *Diff*)

subgoal for *ss3′ ss4′*
apply (*cases ss3*, *cases ss4*, *cases ss1*, *cases ss2*)
apply (*cases ss3′*, *cases ss4′*, *clarsimp*)
using Δ*3 finals* **unfolding** *ss* **apply** *clarsimp*
apply(*simp-all add*: Δ*3-defs sstatA′-def*)
apply(*cases statO*, *simp-all*) **by** (*cases statA*, *simp-all add*: *newStat-EqI*) **.**

have *vs3 xx* = *vs4 xx* **using** Δ*3 lcfgs* **unfolding** *ss* Δ*3-defs* **apply** *clarsimp*
by (*metis cfgs-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 Δ*3 lcfgs* **unfolding** *ss* Δ*3-defs array-loc-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 cfgs3*) = *7* ⟶ *oor* Δ*3* Δ*1* ∞ *2 2 ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*)
∧ (*pcOf* (*last cfgs3*) = *8* ⟶ *oor* Δ*3* Δ*1* ∞ *1 1 ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*)⟹
∃ *w1′*<*w1*. ∃ *w2′*<*w2*. *oor* Δ*3* Δ*1* ∞ *w1′ w2′ ss3′ ss4′* (*sstatA′*
statA ss3 ss4) *ss1 ss2 statO*
subgoal for *ss3′ ss4′* **apply**(*cases ss3′*, *cases ss4′*)
subgoal for *pstate3′ cfg3′ cfgs3′ ib3′ ls3′*
pstate4′ cfg4′ cfgs4′ ib4′ ls4′
using *lpc3* **apply**(*elim disjE*)

subgoal apply(*intro exI*[*of* - *2*], *intro conjI*)
subgoal using Δ*3* **unfolding** *ss* Δ*3-defs* **apply** *clarify*
by (*metis enat-ord-simps*(*4*) *numeral-ne-infinity*)
apply(*intro exI*[*of* - *2*], *rule conjI*)
subgoal using Δ*3* **unfolding** *ss* Δ*3-defs* **apply** *clarify*
by (*metis enat-ord-simps*(*4*) *numeral-ne-infinity*)
by *simp*

subgoal apply(*intro exI*[*of* - *1*], *intro conjI*)
subgoal using Δ*3* **unfolding** *ss* Δ*3-defs* **apply** *clarify*
by (*metis one-less-numeral-iff semiring-norm*(*76*))
apply(*intro exI*[*of* - *1*], *rule conjI*)
subgoal using Δ*3* **unfolding** *ss* Δ*3-defs* **apply** *clarify*
by (*metis one-less-numeral-iff semiring-norm*(*76*))
by *simp* **. . .**

show *react* (*oor* Δ*3* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* (*oor* Δ*3* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* (*oor* Δ*3* Δ*1*) *w1 w2 ss3 ss4 statA ss1 ss2 statO*

unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* (*oor* Δ*3* Δ*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' 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* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*3* *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
next
case *spec-resolve*
then show *?thesis*
using Δ*3* *prem*(*6*) *resolve-127*
unfolding *ss* Δ*3-defs* **by** (*clarify*,*metis cfgs-map misSpecL1-def*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*3* *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
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* Δ*3* *lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3* *lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*3* *lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*, *metis config.sel*(*1*) *last-map*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*3* *lcfgs sn3* **unfolding** *ss*
by (*simp add*: Δ*3-defs*, *metis config.sel*(*1*) *last-map*)

next
case *spec-resolve*
then show *?thesis* **using** *stat* Δ*3 lcfgs sn3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-normal* **note** *sn4* = *spec-normal*
show *?thesis*
apply(*intro oorI1*)
unfolding *ss* Δ*3-def prem*(*4*,*5*) **apply**− **apply**(*clarify*,*intro conjI*)
subgoal using *stat* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss prem*(*4*,*5*) *hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp-all add*: Δ*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* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss prem*(*4*,*5*) *hh*
apply− **apply**(*frule* Δ*3-implies*) **by**(*simp add*: Δ*3-defs*)
subgoal using *stat* Δ*3 lcfgs prem*(*1*,*2*) *sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*3-defs*)
using *cases-14*[*of pc3*] **apply** *simp* **apply**(*elim disjE*)
by *simp-all*
subgoal using *stat* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **apply**(*simp add*: Δ*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* Δ*3 lcfgs sn3 sn4* **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **by**(*simp add*: Δ*3-defs*)
subgoal using *stat* Δ*3 lcfgs sn3 sn4 prem*(*6*) **unfolding** *ss hh*
apply− **apply**(*frule* Δ*3-implies*) **by**(*simp add*: Δ*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** *stat* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*3* **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)

next
case *spec-mispred*
then show *?thesis* **using** *stat* $\Delta 3$ *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* $\Delta 3$ *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* $\Delta 3$ *prem*(*6*) **unfolding** *ss* **by** (*auto simp add*: Δ*3-defs*)
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*: Δ*3-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* $\Delta 3$ *lcfgs sr3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* $\Delta 3$ *lcfgs sr3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* $\Delta 3$ *lcfgs sr3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* $\Delta 3$ *lcfgs sr3* **unfolding** *ss* **by** (*simp add*: Δ*3-defs*)
next
case *spec-resolve* **note** *sr4* = *spec-normal*
show *?thesis* **using** *stat* $\Delta 3$ *prem sr3 sr4*
unfolding *ss lcfgs* **apply**−
apply(*frule* Δ*3-implies*) **apply** (*simp add*: Δ*3-defs* Δ*1-defs*)
apply(*rule oorI2*, *simp add*: Δ*1-defs local.spec-resolve*)
by (*metis prem*(*1*) *ss3*)
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 *Δ1′*: *Δ1′ 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 *Δ1′* **unfolding** *ss Δ1′-def*
  apply *clarsimp*
  by(*frule common-implies*, *simp*)

  have *f2*:¬*finalN ss2*
  using *Δ1′* **unfolding** *ss Δ1′-def*
  apply *clarsimp*
  by(*frule common-implies*, *simp*)

  have *f3*:¬*finalS ss3*
    using *Δ1′* **unfolding** *ss*
    apply−**apply**(*frule Δ1′-implies*)
    by (*simp add*: *finalS-while-spec*)

  have *f4*:¬*finalS ss4*
    using *Δ1′* **unfolding** *ss*
    apply−**apply**(*frule Δ1′-implies*)
    by (*simp add*: *finalS-while-spec*)

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 *match12-aux*:
(⋀*s1′ s2′ statA′*.
statA′ = *sstatA′ statA ss3 ss4* ⟹
validTransO (*ss3*, *s1′*) ⟹
validTransO (*ss4*, *s2′*) ⟹
Opt.eqAct ss3 ss4 ⟹
(¬ *isSecO ss3* ∧ ¬ *isSecO ss4* ∧
(*statA* = *statA′* ∨ *statO* = *Diff*) ∧
Δ*1* ∞ *1 1 s1′ s2′ statA′ ss1 ss2 statO*))
⟹ *match12* Δ*1 w1 w2 ss3 ss4 statA ss1 ss2 statO*
apply(*rule match12-simpleI*, *rule disjI1*)

apply(*rule exI*[*of - 1*], *rule conjI*)
subgoal using Δ*1′* **unfolding** *ss* Δ*1′-defs* **apply** *clarify*
by(*metis enat-ord-simps*(*4*) *infinity-ne-i1*)
apply(*rule exI*[*of - 1*], *rule conjI*)
subgoal using Δ*1′* **unfolding** *ss* Δ*1′-defs* **apply** *clarify*
by(*metis enat-ord-simps*(*4*) *infinity-ne-i1*)
by *auto*

show *react* Δ*1 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*1 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*1 w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*1 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 ¬ isSecO ss3
      using v sa Δ1' unfolding ss
      by (simp add: Δ1'-defs,metis list.size(3) n-not-Suc-n)

    show ¬ isSecO ss4
      using v sa Δ1' unfolding ss
      by (simp add: Δ1'-defs,metis list.size(3) n-not-Suc-n)

  show stat: statA = statA' ∨ statO = Diff

    using v sa Δ1'
    apply (cases ss3, cases ss4, cases ss1, cases ss2)
    apply (cases ss3', cases ss4', clarsimp)
    using v sa Δ1' unfolding ss statA' apply clarsimp
    apply(simp-all add: Δ1'-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)

    show Δ1 ∞ 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 Δ1' stat unfolding ss by (simp add:
Δ1'-defs)
      next
        case nonspec-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' unfolding ss
          apply (simp add: Δ1'-defs, clarify, elim disjE)
          by (simp-all add: Δ1-defs Δ1'-defs)
      next
        case spec-mispred
        then show ?thesis using sa Δ1' unfolding ss
          apply (simp add: Δ1'-defs, clarify, elim disjE)
          by (simp-all add: Δ1-defs Δ1'-defs)
      next
        case spec-normal note sn3 = spec-normal
        show ?thesis using Δ1' sn3(2) unfolding ss
```

apply (*simp add*: Δ*1'-defs*, *clarsimp*)
by (*smt* (*z3*) *insert-commute*)
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** Δ*1' sr3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** Δ*1' sr3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** Δ*1' sr3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*, *metis*)
next
case *spec-normal*
then show *?thesis* **using** Δ*1' sr3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*, *metis*)
next
case *spec-Fence*
then show *?thesis* **using** Δ*1' sr3* **unfolding** *ss* **by** (*simp add*: Δ*1'-defs*, *metis*)
next
case *spec-resolve* **note** *sr4* = *spec-resolve*
show *?thesis*
using *sa stat* Δ*1' v3 v4 sr3 sr4* **unfolding** *ss hh*
apply(*simp add*: Δ*1'-defs* Δ*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* Δ*2'* Δ*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 Δ*2'*: Δ*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 Δ*2'* **unfolding** *ss* Δ*2'-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f2*:¬*finalN ss2*
using Δ*2'* **unfolding** *ss* Δ*2'-def*
apply *clarsimp*
by(*frule common-implies*, *simp*)

have *f3*:¬*finalS ss3*
using Δ*2'* **unfolding** *ss*
apply−**apply**(*frule* Δ*2'-implies*)
using *finalS-while-spec-L2* **by** *force*

have *f4*:¬*finalS ss4*
using Δ*2'* **unfolding** *ss*
apply−**apply**(*frule* Δ*2'-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 $\Delta 2'$ **unfolding** *ss*
by (*simp add*: $\Delta 2'$*-defs*, *metis list.size*(*3*) *zero-neq-numeral*)

have *sec4*[*simp*]:¬ *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* ⟹
validTransO (*ss3*, *s3'*) ⟹ *validTransO* (*ss4*, *s4'*) ⟹
(*statA* = *statA'* ∨ *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' cfgs3' ib3' ibUT3' ls3'*
pstate4' cfg4' cfgs4' ib4' ibUT4' ls4' statA'.
(*pstate3*, *cfg3*, *cfgs3*, *ibT3*, *ibUT3*, *ls3*) →*S* (*pstate3'*, *cfg3'*, *cfgs3'*, *ib3'*, *ibUT3'*, *ls3'*) ⟹
(*pstate4*, *cfg4*, *cfgs4*, *ibT4*, *ibUT4*, *ls4*) →*S* (*pstate4'*, *cfg4'*, *cfgs4'*, *ib4'*, *ibUT4'*, *ls4'*) ⟹
Opt.eqAct ss3 ss4 ⟹ *statA'* = *sstatA' statA ss3 ss4* ⟹
($\Delta 2$ ∞ *1 1* (*pstate3'*, *cfg3'*, *cfgs3'*, *ib3'*, *ibUT3'*, *ls3'*) (*pstate4'*, *cfg4'*, *cfgs4'*, *ib4'*, *ibUT4'*, *ls4'*) *statA' ss1 ss2 statO*))
⟹ *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* **. .**

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* Δ*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* Δ*2'* **unfolding** *ss* **by** (*auto simp add*: Δ*2'-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2'* **unfolding** *ss* **by** (*auto simp add*: Δ*2'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2' prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2'-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*2' prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2'-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2' prem* **unfolding** *ss* **by** (*auto simp add*: Δ*2'-defs*)
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* Δ*2' sr3* **unfolding** *ss* **by** (*simp add*: Δ*2'-defs*)
next
case *nonspec-mispred*
then show *?thesis* **using** *stat* Δ*2' sr3* **unfolding** *ss* **by** (*simp add*: Δ*2'-defs*)
next
case *spec-mispred*
then show *?thesis* **using** *stat* Δ*2' sr3* **unfolding** *ss* **by** (*simp add*: Δ*2'-defs*)
next
case *spec-normal*
then show *?thesis* **using** *stat* Δ*2' sr3* **unfolding** *ss* **by** (*simp add*: Δ*2'-defs*)
next
case *spec-Fence*
then show *?thesis* **using** *stat* Δ*2' sr3* **unfolding** *ss* **by** (*simp add*: Δ*2'-defs*)
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)
        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))

         by (metis map-L2 butlast.simps(2) last.simps last-map list.simps(8)
           map-butlast image-constant-conv not-Cons-self2 image-subset-iff
           list.set-intros(1,2) list.simps(15) resolve.simps resolve-127
           set-empty2 subset-insertI resolve-73 numeral-eq-iff )+
      qed
    qed .
  qed
qed


lemma stepe: unwindIntoCond Δe 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 Δe: Δe 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* ∧ *finalN ss1* = *finalS ss3* ∧ *finalN ss2* = *finalS ss4*
using Δ*e Opt.final-def finalS-def stepS-endPC endPC-def finalB-endPC*
unfolding Δ*e-defs ss* **by** *clarsimp*

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

show *react* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *react-def* **proof**(*intro conjI*)

show *match1* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match1-def* **by** (*simp add*: *finalS-def final-def*)
show *match2* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
unfolding *match2-def* **by** (*simp add*: *finalS-def final-def*)
show *match12* Δ*e w1 w2 ss3 ss4 statA ss1 ss2 statO*
apply(*rule match12-simpleI*) **using** Δ*e* **unfolding** *ss*
by (*simp add*: Δ*e-defs stepS-endPC*)
qed
qed

lemmas *theConds* = *step0 step1 step2 step3 step4 step5 stepe*

proposition *lrsecure*
proof−
define *m* **where** *m*: *m* ≡ (*7*::*nat*)
define Δ*s* **where** Δ*s*: Δ*s* ≡ λ*i*::*nat*.
if i = *0 then* Δ*0*
else if i = *1 then* Δ*1*
else if i = *2 then* Δ*2*
else if i = *3 then* Δ*3*
else if i = *4 then* Δ*1'*
else if i = *5 then* Δ*2'*
else Δ*e*
define *nxt* **where** *nxt*: *nxt* ≡ λ*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* 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 oor5-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.