

# Extension of Stateful Intransitive Noninterference with Inputs, Outputs, and Nondeterminism in Language IMP

Pasquale Noce

Senior Staff Firmware Engineer at HID Global, Italy  
pasquale dot noce dot lavoro at gmail dot com  
pasquale dot noce at hidglobal dot com

February 6, 2026

## Abstract

In a previous paper of mine, the notion of termination-sensitive information flow security with respect to a level-based interference relation, as studied by Volpano, Smith, and Irvine and formalized in Nipkow and Klein’s book on formal programming language semantics (in the version of February 2023), is generalized to the notion of termination-sensitive information flow correctness with respect to an interference function mapping program states to (generally) intransitive interference relations.

This paper extends both the aforesaid information flow correctness criterion and the related static type system to the case of an imperative programming language supporting inputs, outputs, and nondeterminism. Regarding inputs and nondeterminism, Volpano, Smith, and Irvine observe that their soundness theorem no longer holds if their core language is extended with these features. This paper shows that the difficulty can be solved by extending the inductive definition of the language’s operational semantics, which enables to apply a suitably extended information flow correctness criterion based on stateful intransitive noninterference, as well as an extended static type system enforcing this criterion, to such an extended programming language. Although an extension with inputs, outputs, and nondeterminism of the didactic programming language IMP employed in the book is used for this purpose, the introduced concepts apply to larger, real-world imperative programming languages as well.

## Contents

- 1 Extension of language IMP with inputs, outputs, and non-determinism**

**2**

1.1	Extended syntax . . . . .	3
1.2	Extended big-step semantics . . . . .	4
1.3	Extended small-step semantics . . . . .	6
1.4	Equivalence of big-step and small-step semantics . . . . .	7
<b>2</b>	<b>Underlying concepts and formal definitions</b>	<b>8</b>
2.1	Global context definitions . . . . .	8
2.2	Local context definitions . . . . .	10
<b>3</b>	<b>Idempotence of the auxiliary type system meant for loop bodies</b>	<b>23</b>
3.1	Local context proofs . . . . .	23
<b>4</b>	<b>Overapproximation of program semantics by the main type system</b>	<b>26</b>
4.1	Global context proofs . . . . .	27
4.2	Local context proofs . . . . .	27
<b>5</b>	<b>Sufficiency of well-typedness for information flow correctness: propaedeutic lemmas</b>	<b>37</b>
5.1	Global context proofs . . . . .	37
5.2	Local context proofs . . . . .	46
<b>6</b>	<b>Sufficiency of well-typedness for information flow correctness: main theorem</b>	<b>64</b>
6.1	Local context proofs . . . . .	64

# 1 Extension of language IMP with inputs, outputs, and nondeterminism

```

theory Small-Step
  imports
    HOL-IMP.BExp
    HOL-IMP.Star
begin

```

In a previous paper of mine [10], the notion of termination-sensitive information flow security with respect to a level-based interference relation, as studied in [12], [11] and formalized in [8], is generalized to the notion of termination-sensitive information flow correctness with respect to an interference function mapping program states to (generally) intransitive interference relations. Moreover, a static type system is specified and is proven to be capable of enforcing such information flow correctness policies.

The present paper extends both the aforesaid information flow correctness criterion and the related static type system to the case of an imperative programming language supporting inputs, outputs, and nondeterminism. Regarding inputs and nondeterminism, [12], section 7.1, observes that “if we try to extend the core language with a primitive random number generator  $rand()$  and allow an assignment such as  $z := rand()$  to be well typed when  $z$  is low, then the soundness theorem no longer holds”, and from this infers that “new security models [...] should be explored as potential notions of type soundness for new type systems that deal with nondeterministic programs”. The present paper shows that this difficulty can be solved by extending the inductive definition of the programming language’s operational semantics so as to reflect the fact that, even though the input instruction  $z := rand()$  may set  $z$  to an arbitrary input value, the same program state is produced whenever the input value is the same. As shown in this paper, this enables to apply a suitably extended information flow correctness criterion based on stateful intransitive noninterference, as well as an extended static type system enforcing this criterion, to such an extended programming language. The didactic imperative programming language IMP employed in [8], extended with an input instruction, an output instruction, and a control structure allowing for nondeterministic choice, will be used for this purpose. Yet, in the same way as in my previous paper [10], the introduced concepts are applicable to larger, real-world imperative programming languages, too, by just affording the additional type system complexity arising from richer language constructs.

For further information about the formal definitions and proofs contained in this paper, refer to Isabelle documentation, particularly [9], [4], [2], [3], and [1].

As mentioned above, the first task to be tackled, which is the subject of this section, consists of extending the original syntax, big-step operational semantics, and small-step operational semantics of language IMP, as formalized in [6], [5], and [7], respectively.

## 1.1 Extended syntax

The starting point is extending the original syntax of language IMP with the following additional constructs.

- An input instruction  $IN\ x$ , which sets variable  $x$  to an input value.
- An output instruction  $OUT\ x$ , which outputs the current value of variable  $x$ .
- A control structure  $c_1\ OR\ c_2$ , which allows for a nondeterministic choice between commands  $c_1$  and  $c_2$ .

**declare**  $[[\text{syntax-ambiguity-warning} = \text{false}]]$

**datatype** *com* =  
  *SKIP* |  
  *Assign vname aexp* ( $\langle \text{-} ::= \text{-} \rangle [1000, 61] 70$ ) |  
  *Input vname* ( $\langle \text{(IN -)} \rangle [61] 70$ ) |  
  *Output vname* ( $\langle \text{(OUT -)} \rangle [61] 70$ ) |  
  *Seq com com* ( $\langle \text{-}; \text{-} \rangle [61, 61] 70$ ) |  
  *Or com com* ( $\langle \text{- OR -} \rangle [61, 61] 70$ ) |  
  *If bexp com com* ( $\langle \text{(IF -/ THEN -/ ELSE -)} \rangle [0, 0, 61] 70$ ) |  
  *While bexp com* ( $\langle \text{(WHILE -/ DO -)} \rangle [0, 61] 70$ )

## 1.2 Extended big-step semantics

The original big-step semantics of language IMP associates a pair formed by a command and an initial *program execution stage*, consisting of a program state, with a corresponding final program execution stage, consisting of a program state as well. The extended big-step semantics defined here below extends such program execution stage notion by considering, in addition to a program state, the following additional parameters.

- A *stream of input values*, consisting of a function  $f$  mapping each pair formed by a variable and a natural number with an integer value, where  $f x n$  is the input value assigned to variable  $x$  by an input instruction  $IN x$  after  $n$  previous such assignments to  $x$ .
- A *trace of inputs*, consisting of a list  $vs$  of pairs formed by a variable and an integer value, to which a further element  $(x, i)$  is appended as a result of the execution of an input instruction  $IN x$ , where  $i$  is the input value assigned to variable  $x$ .
- A *trace of outputs*, consisting of a list  $ws$  of pairs formed by a variable and an integer value, to which a further element  $(x, i)$  is appended as a result of the execution of an output instruction  $OUT x$ , where  $i$  is the current value of variable  $x$  being output.

Unlike the other components of a program execution stage, the stream of input values is an *invariant* of the big-step semantics, and then also of the small-step semantics defined subsequently, in that any two program execution stages associated with each other by either semantics share the same stream of input values.

**type-synonym**  $stream = vname \Rightarrow nat \Rightarrow val$

**type-synonym**  $inputs = (vname \times val) list$

**type-synonym**  $outputs = (vname \times val) list$

**type-synonym**  $stage = state \times stream \times inputs \times outputs$

**inductive**  $big\text{-}step :: com \times stage \Rightarrow stage \Rightarrow bool$   
**(infix <=> 55) where**  
*Skip:*  
 $(SKIP, p) \Rightarrow p \mid$   
*Assign:*  
 $(x ::= a, s, p) \Rightarrow (s(x := aval\ a\ s), p) \mid$   
*Input:*  
 $n = length\ [p \leftarrow vs.\ fst\ p = x] \Longrightarrow (IN\ x, s, f, vs, ws) \Rightarrow$   
 $(s(x := f\ x\ n), f, vs\ @\ [(x, f\ x\ n)], ws) \mid$   
*Output:*  
 $(OUT\ x, s, f, vs, ws) \Rightarrow (s, f, vs, ws\ @\ [(x, s\ x)]) \mid$   
*Seq:*  
 $\llbracket (c_1, p_1) \Rightarrow p_2; (c_2, p_2) \Rightarrow p_3 \rrbracket \Longrightarrow (c_1;;\ c_2, p_1) \Rightarrow p_3 \mid$   
*Or1:*  
 $(c_1, p) \Rightarrow p' \Longrightarrow (c_1\ OR\ c_2, p) \Rightarrow p' \mid$   
*Or2:*  
 $(c_2, p) \Rightarrow p' \Longrightarrow (c_1\ OR\ c_2, p) \Rightarrow p' \mid$   
*IfTrue:*  
 $\llbracket bval\ b\ s; (c_1, s, p) \Rightarrow p' \rrbracket \Longrightarrow$   
 $(IF\ b\ THEN\ c_1\ ELSE\ c_2, s, p) \Rightarrow p' \mid$   
*IfFalse:*  
 $\llbracket \neg\ bval\ b\ s; (c_2, s, p) \Rightarrow p' \rrbracket \Longrightarrow$   
 $(IF\ b\ THEN\ c_1\ ELSE\ c_2, s, p) \Rightarrow p' \mid$   
*WhileFalse:*  
 $\neg\ bval\ b\ s \Longrightarrow (WHILE\ b\ DO\ c, s, p) \Rightarrow (s, p) \mid$   
*WhileTrue:*  
 $\llbracket bval\ b\ s_1; (c, s_1, p_1) \Rightarrow (s_2, p_2);$   
 $(WHILE\ b\ DO\ c, s_2, p_2) \Rightarrow (s_3, p_3) \rrbracket \Longrightarrow$   
 $(WHILE\ b\ DO\ c, s_1, p_1) \Rightarrow (s_3, p_3)$

**declare**  $big\text{-}step.intros\ [intro]$

**inductive-cases**  $SkipE\ [elim!]: (SKIP, p) \Rightarrow p'$

**inductive-cases**  $AssignE\ [elim!]: (x ::= a, p) \Rightarrow p'$

**inductive-cases**  $InputE\ [elim!]: (IN\ x, p) \Rightarrow p'$

**inductive-cases**  $OutputE\ [elim!]: (OUT\ x, p) \Rightarrow p'$

**inductive-cases**  $SeqE\ [elim!]: (c_1;;\ c_2, p) \Rightarrow p'$

**inductive-cases**  $OrE\ [elim!]: (c_1\ OR\ c_2, p) \Rightarrow p'$

**inductive-cases**  $IfE\ [elim!]: (IF\ b\ THEN\ c_1\ ELSE\ c_2, p) \Rightarrow p'$

**inductive-cases** *WhileE* [elim]: (*WHILE* *b DO c*, *p*)  $\Rightarrow$  *p'*

### 1.3 Extended small-step semantics

The original small-step semantics of language IMP associates a pair formed by a command and a program execution stage, which consists of a program state, with another such pair, formed by a command to be executed next and a resulting program execution stage, which consists of a program state as well. The extended small-step semantics defined here below rather uses the same extended program execution stage notion as the extended big-step semantics specified above, and is defined accordingly.

**inductive** *small-step* :: *com*  $\times$  *stage*  $\Rightarrow$  *com*  $\times$  *stage*  $\Rightarrow$  *bool*

(**infix**  $\langle \rightarrow \rangle$  55) **where**

*Assign*:

(*x ::= a*, *s*, *p*)  $\rightarrow$  (*SKIP*, *s*(*x* := *aval a s*), *p*) |

*Input*:

*n* = *length* [*p*  $\leftarrow$  *vs*. *fst p* = *x*]  $\Longrightarrow$  (*IN* *x*, *s*, *f*, *vs*, *ws*)  $\rightarrow$   
 (*SKIP*, *s*(*x* := *f x n*), *f*, *vs* @ [(*x*, *f x n*)], *ws*) |

*Output*:

(*OUT* *x*, *s*, *f*, *vs*, *ws*)  $\rightarrow$  (*SKIP*, *s*, *f*, *vs*, *ws* @ [(*x*, *s x*)] |

*Seq1*:

(*SKIP*;; *c*<sub>2</sub>, *p*)  $\rightarrow$  (*c*<sub>2</sub>, *p*) |

*Seq2*:

(*c*<sub>1</sub>, *p*)  $\rightarrow$  (*c*<sub>1</sub>', *p*')  $\Longrightarrow$  (*c*<sub>1</sub>;; *c*<sub>2</sub>, *p*)  $\rightarrow$  (*c*<sub>1</sub>';; *c*<sub>2</sub>, *p*') |

*Or1*:

(*c*<sub>1</sub> *OR* *c*<sub>2</sub>, *p*)  $\rightarrow$  (*c*<sub>1</sub>, *p*) |

*Or2*:

(*c*<sub>1</sub> *OR* *c*<sub>2</sub>, *p*)  $\rightarrow$  (*c*<sub>2</sub>, *p*) |

*IfTrue*:

*bval b s*  $\Longrightarrow$  (*IF b THEN c*<sub>1</sub> *ELSE c*<sub>2</sub>, *s*, *p*)  $\rightarrow$  (*c*<sub>1</sub>, *s*, *p*) |

*IfFalse*:

$\neg$  *bval b s*  $\Longrightarrow$  (*IF b THEN c*<sub>1</sub> *ELSE c*<sub>2</sub>, *s*, *p*)  $\rightarrow$  (*c*<sub>2</sub>, *s*, *p*) |

*WhileFalse*:

$\neg$  *bval b s*  $\Longrightarrow$  (*WHILE b DO c*, *s*, *p*)  $\rightarrow$  (*SKIP*, *s*, *p*) |

*WhileTrue*:

*bval b s*  $\Longrightarrow$  (*WHILE b DO c*, *s*, *p*)  $\rightarrow$  (*c*;; *WHILE b DO c*, *s*, *p*)

**declare** *small-step.intros* [*simp*, *intro*]

**inductive-cases** *skipE* [elim!]: (*SKIP*, *p*)  $\rightarrow$  *cf*

**inductive-cases** *assignE* [elim!]: (*x ::= a*, *p*)  $\rightarrow$  *cf*

**inductive-cases** *inputE* [elim!]: (*IN* *x*, *p*)  $\rightarrow$  *cf*

**inductive-cases** *outputE* [elim!]: (*OUT* *x*, *p*)  $\rightarrow$  *cf*

**inductive-cases** *seqE* [elim!]: (*c*<sub>1</sub>;; *c*<sub>2</sub>, *p*)  $\rightarrow$  *cf*

**inductive-cases** *orE* [elim!]: (*c*<sub>1</sub> *OR* *c*<sub>2</sub>, *p*)  $\rightarrow$  *cf*

**inductive-cases** *ifE* [elim!]: (*IF* *b* *THEN* *c*<sub>1</sub> *ELSE* *c*<sub>2</sub>, *p*)  $\rightarrow$  *cf*

**inductive-cases** *whileE* [elim]: (*WHILE* *b* *DO* *c*, *p*)  $\rightarrow$  *cf*

**abbreviation** *small-steps* :: *com*  $\times$  *stage*  $\Rightarrow$  *com*  $\times$  *stage*  $\Rightarrow$  *bool*

(**infix**  $\langle \rightarrow^* \rangle$  55) **where**

*cf*  $\rightarrow^*$  *cf'*  $\equiv$  *star small-step cf cf'*

**function** *small-steps1* ::

*com*  $\times$  *stage*  $\Rightarrow$  (*com*  $\times$  *stage*) *list*  $\Rightarrow$  *com*  $\times$  *stage*  $\Rightarrow$  *bool*

( $\langle (- \rightarrow^* \{-'\} -) \rangle$  [51, 51] 55)

**where**

*cf*  $\rightarrow^* \{\ [] \}$  *cf'* = (*cf* = *cf'*) |

*cf*  $\rightarrow^* \{ cfs @ [cf'] \}$  *cf''* = (*cf*  $\rightarrow^* \{ cfs \}$  *cf'*  $\wedge$  *cf'*  $\rightarrow$  *cf''*)

$\langle$ *proof* $\rangle$

**termination**  $\langle$ *proof* $\rangle$

## 1.4 Equivalence of big-step and small-step semantics

**lemma** *star-seq2*:

(*c*<sub>1</sub>, *p*)  $\rightarrow^*$  (*c*<sub>1</sub>', *p'*)  $\implies$  (*c*<sub>1</sub>;; *c*<sub>2</sub>, *p*)  $\rightarrow^*$  (*c*<sub>1</sub>';; *c*<sub>2</sub>, *p'*)

$\langle$ *proof* $\rangle$

**lemma** *seq-comp*:

$\llbracket (c_1, p_1) \rightarrow^* (SKIP, p_2); (c_2, p_2) \rightarrow^* (SKIP, p_3) \rrbracket \implies$   
(*c*<sub>1</sub>;; *c*<sub>2</sub>, *p*<sub>1</sub>)  $\rightarrow^*$  (*SKIP*, *p*<sub>3</sub>)

$\langle$ *proof* $\rangle$

**lemma** *big-to-small*:

*cf*  $\Rightarrow$  *p*  $\implies$  *cf*  $\rightarrow^*$  (*SKIP*, *p*)

$\langle$ *proof* $\rangle$

**lemma** *small1-big-continue*:

$\llbracket cf \rightarrow cf'; cf' \Rightarrow p \rrbracket \implies cf \Rightarrow p$

$\langle$ *proof* $\rangle$

**lemma** *small-to-big*:

*cf*  $\rightarrow^*$  (*SKIP*, *p*)  $\implies cf \Rightarrow p$

$\langle$ *proof* $\rangle$

**lemma** *big-iff-small*:

```

  cf  $\Rightarrow$  p = cf  $\rightarrow^*$  (SKIP, p)
<proof>

```

```

end

```

## 2 Underlying concepts and formal definitions

```

theory Definitions
  imports Small-Step
begin

```

### 2.1 Global context definitions

As compared with my previous paper [10]:

- Type *flow*, which models any potential program execution flow as a list of instructions, occurring in their order of execution, is extended with two additional instructions, namely an input instruction *IN*  $x$  and an output instruction *OUT*  $x$  standing for the respective additional commands of the considered extension of language IMP.
- Function *run-flow*, which used to map a pair formed by such a program execution flow  $cs$  and a starting program state  $s$  to the resulting program state, here takes two additional parameters, namely a starting trace of inputs  $vs$  and a stream of input values  $f$ , since they are required as well for computing the resulting program state according to the semantics of the considered extension of language IMP.

```

declare [[syntax-ambiguity-warning = false]]

```

```

datatype com-flow =
  Assign vname aexp (<- ::= -> [1000, 61] 70) |
  Input vname (<(IN -)> [61] 70) |
  Output vname (<(OUT -)> [61] 70) |
  Observe vname set (<<-> [61] 70)

```

```

type-synonym flow = com-flow list
type-synonym tag = vname  $\times$  nat
type-synonym config = state set  $\times$  vname set
type-synonym scope = config set  $\times$  bool
type-synonym state-upd = vname  $\times$  val option

```

```

definition eq-streams ::
  stream  $\Rightarrow$  stream  $\Rightarrow$  inputs  $\Rightarrow$  inputs  $\Rightarrow$  tag set  $\Rightarrow$  bool

```

$\langle \langle (- = -'(\subseteq -, -, -')) \rangle [51, 51] 50 \rangle$  **where**  
 $f = f' (\subseteq vs, vs', T) \equiv \forall (x, n) \in T.$   
 $f x (\text{length } [p \leftarrow vs. \text{fst } p = x] + n) =$   
 $f' x (\text{length } [p \leftarrow vs'. \text{fst } p = x] + n)$

**abbreviation**  $eq\text{-states} :: state \Rightarrow state \Rightarrow \text{vname set} \Rightarrow bool$   
 $\langle \langle (- = -'(\subseteq -')) \rangle [51, 51] 50 \rangle$  **where**  
 $s = t (\subseteq X) \equiv \forall x \in X. s x = t x$

**abbreviation**  $univ\text{-states} :: state\ set \Rightarrow \text{vname set} \Rightarrow state\ set$   
 $\langle \langle (Univ -'(\subseteq -')) \rangle [51] 75 \rangle$  **where**  
 $Univ\ A (\subseteq X) \equiv \{s. \exists t \in A. s = t (\subseteq X)\}$

**abbreviation**  $univ\text{-vars-if} :: state\ set \Rightarrow \text{vname set} \Rightarrow \text{vname set}$   
 $\langle \langle (Univ?? - -) \rangle [51, 75] 75 \rangle$  **where**  
 $Univ??\ A\ X \equiv \text{if } A = \{\} \text{ then } UNIV \text{ else } X$

**abbreviation**  $tl2\ xs \equiv tl (tl\ xs)$

**primrec**  $avars :: aexp \Rightarrow \text{vname set where}$   
 $avars (N\ i) = \{\}$  |  
 $avars (V\ x) = \{x\}$  |  
 $avars (Plus\ a_1\ a_2) = avars\ a_1 \cup avars\ a_2$

**primrec**  $bvars :: bexp \Rightarrow \text{vname set where}$   
 $bvars (Bc\ v) = \{\}$  |  
 $bvars (Not\ b) = bvars\ b$  |  
 $bvars (And\ b_1\ b_2) = bvars\ b_1 \cup bvars\ b_2$  |  
 $bvars (Less\ a_1\ a_2) = avars\ a_1 \cup avars\ a_2$

**fun**  $no\text{-upd} :: flow \Rightarrow \text{vname set} \Rightarrow bool$  **where**  
 $no\text{-upd } (x ::= - \# cs)\ X = (x \notin X \wedge no\text{-upd } cs\ X)$  |  
 $no\text{-upd } (IN\ x \# cs)\ X = (x \notin X \wedge no\text{-upd } cs\ X)$  |  
 $no\text{-upd } (OUT\ x \# cs)\ X = (x \notin X \wedge no\text{-upd } cs\ X)$  |  
 $no\text{-upd } (- \# cs)\ X = no\text{-upd } cs\ X$  |  
 $no\text{-upd } - = True$

**fun**  $flow\text{-aux} :: com\ list \Rightarrow flow$  **where**  
 $flow\text{-aux } (x ::= a \# cs) = (x ::= a) \# flow\text{-aux } cs$  |  
 $flow\text{-aux } (IN\ x \# cs) = IN\ x \# flow\text{-aux } cs$  |  
 $flow\text{-aux } (OUT\ x \# cs) = OUT\ x \# flow\text{-aux } cs$  |  
 $flow\text{-aux } (IF\ b\ THEN\ -\ ELSE\ - \# cs) = \langle bvars\ b \rangle \# flow\text{-aux } cs$  |  
 $flow\text{-aux } (WHILE\ b\ DO\ - \# cs) = \langle bvars\ b \rangle \# flow\text{-aux } cs$  |  
 $flow\text{-aux } (c;; - \# cs) = flow\text{-aux } (c \# cs)$  |  
 $flow\text{-aux } (- \# cs) = flow\text{-aux } cs$  |  
 $flow\text{-aux } [] = []$

**definition**  $flow :: (com \times stage) list \Rightarrow flow$  **where**  
 $flow\ cfs = flow\ aux\ (map\ fst\ cfs)$

**function**  $in\ flow :: flow \Rightarrow inputs \Rightarrow stream \Rightarrow inputs$  **where**  
 $in\ flow\ (cs\ @\ [-\ ::=-])\ vs\ f = in\ flow\ cs\ vs\ f\ |$   
 $in\ flow\ (cs\ @\ [IN\ x])\ vs\ f = in\ flow\ cs\ vs\ f\ @\ (let$   
 $\quad n = length\ [p \leftarrow vs.\ fst\ p = x] + length\ [c \leftarrow cs.\ c = IN\ x]$   
 $\quad in\ [(x,\ f\ x\ n)])\ |$   
 $in\ flow\ (cs\ @\ [OUT\ -])\ vs\ f = in\ flow\ cs\ vs\ f\ |$   
 $in\ flow\ (cs\ @\ [(-)])\ vs\ f = in\ flow\ cs\ vs\ f\ |$   
 $in\ flow\ []\ -\ - = []$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

**function**  $run\ flow :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow state$  **where**  
 $run\ flow\ (cs\ @\ [x\ ::= a])\ vs\ s\ f = (let\ t = run\ flow\ cs\ vs\ s\ f$   
 $\quad in\ t(x := aval\ a\ t))\ |$   
 $run\ flow\ (cs\ @\ [IN\ x])\ vs\ s\ f = (let\ t = run\ flow\ cs\ vs\ s\ f;$   
 $\quad n = length\ [p \leftarrow vs.\ fst\ p = x] + length\ [c \leftarrow cs.\ c = IN\ x]$   
 $\quad in\ t(x := f\ x\ n))\ |$   
 $run\ flow\ (cs\ @\ [OUT\ -])\ vs\ s\ f = run\ flow\ cs\ vs\ s\ f\ |$   
 $run\ flow\ (cs\ @\ [(-)])\ vs\ s\ f = run\ flow\ cs\ vs\ s\ f\ |$   
 $run\ flow\ []\ vs\ s\ - = s$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

**function**  $out\ flow :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow outputs$  **where**  
 $out\ flow\ (cs\ @\ [-\ ::=-])\ vs\ s\ f = out\ flow\ cs\ vs\ s\ f\ |$   
 $out\ flow\ (cs\ @\ [IN\ -])\ vs\ s\ f = out\ flow\ cs\ vs\ s\ f\ |$   
 $out\ flow\ (cs\ @\ [OUT\ x])\ vs\ s\ f = (let\ t = run\ flow\ cs\ vs\ s\ f$   
 $\quad in\ out\ flow\ cs\ vs\ s\ f\ @\ [(x,\ t\ x)])\ |$   
 $out\ flow\ (cs\ @\ [(-)])\ vs\ s\ f = out\ flow\ cs\ vs\ s\ f\ |$   
 $out\ flow\ []\ -\ -\ - = []$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

## 2.2 Local context definitions

**locale**  $noninterf =$   
**fixes**

```

interf :: state => 'd => 'd => bool
  (<(-: - ~> -)> [51, 51, 51] 50) and
dom :: vname => 'd and
state :: vname set
assumes
  interf-state: s = t ( $\subseteq$  state)  $\implies$  interf s = interf t

```

**context** *noninterf*  
**begin**

As in my previous paper [10], function *sources* is defined along with an auxiliary function *sources-ax* by means of mutual recursion. According to this definition, the set of variables *sources cs vs s f x*, where:

- *cs* is a program execution flow,
- *vs* is a trace of inputs,
- *s* is a program state,
- *f* is a stream of input values, and
- *x* is a variable,

contains a variable *y* if there exist a descending sequence of left sublists  $cs_{n+1}, cs_n @ [c_n], \dots, cs_1 @ [c_1]$  of *cs* and a sequence of variables  $y_{n+1}, \dots, y_1$ , where  $n \geq 1$ ,  $cs_{n+1} = cs$ ,  $y_{n+1} = x$ , and  $y_1 = y$ , satisfying the following conditions.

- For each positive integer  $i \leq n$ , the instruction  $c_i$  is an assignment  $y_{i+1} ::= a_i$  such that:
  - $y_i \in avars a_i$ ,
  - *run-flow*  $cs_i vs s f$ :  $dom y_i \rightsquigarrow dom y_{i+1}$ , and
  - the right sublist of  $cs_{i+1}$  complementary to  $cs_i @ [c_i]$  does not comprise any assignment or input instruction setting variable  $y_{i+1}$  (as the assignment  $c_i$  would otherwise be irrelevant),

or else an observation  $\langle X_i \rangle$  such that:

- $y_i \in X_i$  and
- *run-flow*  $cs_i vs s f$ :  $dom y_i \rightsquigarrow dom y_{i+1}$ .
- The program execution flow  $cs_1$  does not comprise any assignment or input instruction setting variable *y*.

In addition,  $sources\ cs\ vs\ s\ f\ x$  contains variable  $x$  also if the program execution flow  $cs$  does not comprise any assignment or input instruction setting variable  $x$ .

**function**

$sources :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow vname \Rightarrow vname\ set$  **and**  
 $sources\ aux :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow vname \Rightarrow vname\ set$

**where**

$sources\ (cs\ @\ [c])\ vs\ s\ f\ x = (case\ c\ of$   
 $z ::= a \Rightarrow if\ z = x$   
 $\quad then\ sources\ aux\ cs\ vs\ s\ f\ x \cup \bigcup \{sources\ cs\ vs\ s\ f\ y \mid y.$   
 $\quad\quad run\ flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in avars\ a\}$   
 $\quad else\ sources\ cs\ vs\ s\ f\ x \mid$   
 $IN\ z \Rightarrow if\ z = x$   
 $\quad then\ sources\ aux\ cs\ vs\ s\ f\ x$   
 $\quad else\ sources\ cs\ vs\ s\ f\ x \mid$   
 $\langle X \rangle \Rightarrow$   
 $\quad sources\ cs\ vs\ s\ f\ x \cup \bigcup \{sources\ cs\ vs\ s\ f\ y \mid y.$   
 $\quad\quad run\ flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\} \mid$   
 $- \Rightarrow$   
 $\quad sources\ cs\ vs\ s\ f\ x) \mid$

$sources\ [] - - - x = \{x\} \mid$

$sources\ aux\ (cs\ @\ [c])\ vs\ s\ f\ x = (case\ c\ of$   
 $\langle X \rangle \Rightarrow$   
 $\quad sources\ aux\ cs\ vs\ s\ f\ x \cup \bigcup \{sources\ cs\ vs\ s\ f\ y \mid y.$   
 $\quad\quad run\ flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\} \mid$   
 $- \Rightarrow$   
 $\quad sources\ aux\ cs\ vs\ s\ f\ x) \mid$

$sources\ aux\ [] - - - = \{\}$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

**lemmas**  $sources\ induct = sources\ sources\ aux.induct$

Function  $sources\ out$ , defined here below, takes the same parameters  $cs$ ,  $vs$ ,  $s$ ,  $f$ , and  $x$  as function  $sources$ , and returns the set of the variables whose values in the program state  $s$  are allowed to affect the outputs of variable  $x$  possibly occurring as a result of the execution of flow  $cs$  if it starts from the initial state  $s$  and the initial trace of inputs  $vs$ , and takes place according to the stream of input values  $f$ .

In more detail, the set of variables  $sources\ out\ cs\ vs\ s\ f\ x$  is defined as the

union of any set of variables *sources*  $cs_i$  vs  $s f x_i$ , where  $cs_i @ [c_i]$  is any left sublist of  $cs$  such that the instruction  $c_i$  is an output instruction *OUT*  $x$ , in which case  $x_i = x$ , or else an observation  $\langle X_i \rangle$  such that:

- $x_i \in X_i$  and
- *run-flow*  $cs_i$  vs  $s f$ :  $dom x_i \rightsquigarrow dom x$ .

**function**

*sources-out* :: *flow*  $\Rightarrow$  *inputs*  $\Rightarrow$  *state*  $\Rightarrow$  *stream*  $\Rightarrow$  *vname*  $\Rightarrow$  *vname set*

**where**

*sources-out* ( $cs @ [c]$ ) vs  $s f x =$  (case  $c$  of  
*OUT*  $z \Rightarrow$   
*sources-out*  $cs$  vs  $s f x \cup$  (if  $z = x$  then *sources*  $cs$  vs  $s f x$  else  $\{\}$ ) |  
 $\langle X \rangle \Rightarrow$   
*sources-out*  $cs$  vs  $s f x \cup \bigcup \{sources\ cs\ vs\ s\ f\ y\ | y.$   
*run-flow*  $cs$  vs  $s f$ :  $dom y \rightsquigarrow dom x \wedge y \in X\}$  |  
 $- \Rightarrow$   
*sources-out*  $cs$  vs  $s f x)$  |

*sources-out* [] - - - =  $\{\}$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

Function *tags*, defined here below, takes the same parameters  $cs$ ,  $vs$ ,  $s$ ,  $f$ , and  $x$  as the previous functions, and returns the set of the *tags*, namely of the pairs  $(y, m)$  where  $y$  is a variable and  $m$  is a natural number, such that the  $m$ -th input instruction *IN*  $y$  within flow  $cs$  is allowed to affect the value of variable  $x$  resulting from the execution of  $cs$  if it starts from the initial state  $s$  and the initial trace of inputs  $vs$ , and takes place according to the stream of input values  $f$ .

In more detail, the set of tags *tags*  $cs$  vs  $s f x$  contains a tag  $(y, m)$  just in case there exist a descending sequence of left sublists  $cs_{n+1}$ ,  $cs_n @ [c_n]$ , ...,  $cs_1 @ [c_1]$  of  $cs$  and a sequence of variables  $y_{n+1}$ , ...,  $y_1$ , where  $n \geq 1$ ,  $cs_{n+1} = cs$ ,  $y_{n+1} = x$ ,  $y_1 = y$ , and  $y = x$  if  $n = 1$ , satisfying the following conditions.

- For each integer  $i$ , if any, such that  $1 < i \leq n$ , the instruction  $c_i$  is an assignment  $y_{i+1} ::= a_i$  such that:
  - $y_i \in avars\ a_i$ ,
  - *run-flow*  $cs_i$  vs  $s f$ :  $dom y_i \rightsquigarrow dom y_{i+1}$ , and

- the right sublist of  $cs_{i+1}$  complementary to  $cs_i @ [c_i]$  does not comprise any assignment or input instruction setting variable  $y_{i+1}$  (as the assignment  $c_i$  would otherwise be irrelevant),

or else an observation  $\langle X_i \rangle$  such that:

- $y_i \in X_i$  and
- $run-flow\ cs_i\ vs\ s\ f: dom\ y_i \rightsquigarrow dom\ y_{i+1}$ .

- The instruction  $c_1$  is the  $m$ -th input instruction  $IN\ y$  within flow  $cs$ .
- The right sublist of  $cs_2$  complementary to  $cs_1 @ [c_1]$  does not comprise any assignment or input instruction setting variable  $y$  (as the input instruction  $c_1$  would otherwise be irrelevant).

**function**

$tags :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow vname \Rightarrow tag\ set$  **and**

$tags-aux :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow vname \Rightarrow tag\ set$

**where**

$tags\ (cs\ @\ [c])\ vs\ s\ f\ x = (case\ c\ of$

$z ::= a \Rightarrow if\ z = x$

$then\ tags-aux\ cs\ vs\ s\ f\ x \cup \cup \{tags\ cs\ vs\ s\ f\ y \mid y.$

$run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in avars\ a\}$

$else\ tags\ cs\ vs\ s\ f\ x \mid$

$IN\ z \Rightarrow if\ z = x$

$then\ insert\ (x,\ length\ [c \leftarrow cs.\ c = IN\ x])\ (tags-aux\ cs\ vs\ s\ f\ x)$

$else\ tags\ cs\ vs\ s\ f\ x \mid$

$\langle X \rangle \Rightarrow$

$tags\ cs\ vs\ s\ f\ x \cup \cup \{tags\ cs\ vs\ s\ f\ y \mid y.$

$run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\} \mid$

$- \Rightarrow$

$tags\ cs\ vs\ s\ f\ x) \mid$

$tags\ []\ -\ -\ -\ =\ \{\} \mid$

$tags-aux\ (cs\ @\ [c])\ vs\ s\ f\ x = (case\ c\ of$

$\langle X \rangle \Rightarrow$

$tags-aux\ cs\ vs\ s\ f\ x \cup \cup \{tags\ cs\ vs\ s\ f\ y \mid y.$

$run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\} \mid$

$- \Rightarrow$

$tags-aux\ cs\ vs\ s\ f\ x) \mid$

$tags-aux\ []\ -\ -\ -\ =\ \{\}$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

**lemmas**  $tags-induct = tags-tags-aux.induct$

Finally, function  $tags-out$ , defined here below, takes the same parameters  $cs$ ,  $vs$ ,  $s$ ,  $f$ , and  $x$  as the previous functions, and returns the set of the tags  $(y, m)$  such that the  $m$ -th input instruction  $IN\ y$  within flow  $cs$  is allowed to affect the outputs of variable  $x$  possibly occurring as a result of the execution of flow  $cs$  if it starts from the initial state  $s$  and the initial trace of inputs  $vs$ , and takes place according to the stream of input values  $f$ .

In more detail, the set of tags  $tags-out\ cs\ vs\ s\ f\ x$  is defined as the union of any set of tags  $tags\ cs_i\ vs\ s\ f\ x_i$ , where  $cs_i @ [c_i]$  is any left sublist of  $cs$  such that the instruction  $c_i$  is an output instruction  $OUT\ x$ , in which case  $x_i = x$ , or else an observation  $\langle X_i \rangle$  such that:

- $x_i \in X_i$  and
- $run-flow\ cs_i\ vs\ s\ f: dom\ x_i \rightsquigarrow dom\ x$ .

**function**

$tags-out :: flow \Rightarrow inputs \Rightarrow state \Rightarrow stream \Rightarrow vname \Rightarrow tag\ set$

**where**

$tags-out\ (cs\ @\ [c])\ vs\ s\ f\ x = (case\ c\ of$   
   $OUT\ z \Rightarrow$   
     $tags-out\ cs\ vs\ s\ f\ x \cup (if\ z = x\ then\ tags\ cs\ vs\ s\ f\ x\ else\ \{\}) \mid$   
   $\langle X \rangle \Rightarrow$   
     $tags-out\ cs\ vs\ s\ f\ x \cup \bigcup \{tags\ cs\ vs\ s\ f\ y \mid y.$   
       $run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\} \mid$   
   $- \Rightarrow$   
     $tags-out\ cs\ vs\ s\ f\ x) \mid$

$tags-out\ [] - - - = \{\}$

$\langle proof \rangle$

**termination**  $\langle proof \rangle$

Predicate  $correct$ , defined here below, formalizes the extended termination-sensitive information flow correctness criterion. As in my previous paper [10], its parameters consist of a program  $c$ , a set of program states  $A$ , and a set of variables  $X$ .

In more detail, for any state  $s$  agreeing with a state in  $A$  on the value of each state variable contained in  $X$ , let the small-step semantics turn:

- the command  $c$  and the program execution stage  $(s, f, vs, ws)$  into a command  $c_1$  and a program execution stage  $(s_1, f, vs_1, ws_1)$ , and
- the command  $c_1$  and the program execution stage  $(s_1, f, vs_1, ws_1)$  into a command  $c_2$  and a program execution stage  $(s_2, f, vs_2, ws_2)$ .

Furthermore, let:

- $cs$  be the program execution flow leading from  $(c_1, s_1, f, vs_1, ws_1)$  to  $(c_2, s_2, f, vs_2, ws_2)$ , and
- $(t_1, f', vs_1', ws_1')$  be any program execution stage,

and assume that the following conditions hold.

- $S$  is a nonempty subset of the set of the variables  $x$  such that state  $t_1$  agrees with  $s_1$  on the value of each variable contained in *sources*  $cs$   $vs_1$   $s_1$   $f$   $x$ .
- For each variable  $x$  contained in  $S$ , and each tag  $(y, n)$  contained in *tags*  $cs$   $vs_1$   $s_1$   $f$   $x$ , the stream of input values  $f'$  agrees with  $f$  on the input value assigned to variable  $y$  by an input instruction *IN*  $y$  after  $n$  previous such assignments to  $y$  following any one tracked by the starting trace of inputs  $vs_1'$  and  $vs_1$ , respectively.

Then, the information flow is correct only if the small-step semantics turns the command  $c_1$  and the program execution stage  $(t_1, f', vs_1', ws_1')$  into a command  $c_2'$  and a program execution stage  $(t_2, f', vs_2', ws_2')$  satisfying the following correctness conditions.

- $c_2' = SKIP$  just in case  $c_2 = SKIP$ ; namely, program execution terminates just in case it terminates as a result of the execution of flow  $cs$ , so that the two program executions cannot be distinguished based on program termination.
- The resulting sequence of input requests *IN*  $x$  being prompted, where  $x$  is any variable contained in  $S$ , matches the one triggered by the execution of flow  $cs$ , so that the two program executions cannot be distinguished based on those sequences.
- States  $t_2$  and  $s_2$  agree on the value of each variable contained in  $S$ , so that the two program executions cannot be distinguished based on the resulting program states.

Likewise, if the above assumptions hold for functions *sources-out* and *tags-out* in place of functions *sources* and *tags*, respectively, then the information flow correctness requires the first two correctness conditions listed above to hold as well, plus the following one.

- The resulting sequence of outputs of any variable contained in  $S$  matches the one produced by the execution of flow  $cs$ , so that the two program executions cannot be distinguished based on those sequences.

**abbreviation *ok-flow-1* where**

$$\begin{aligned}
\text{ok-flow-1 } c_1 \ c_2 \ c_2' \ s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ cs &\equiv \\
\forall S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } cs \ vs_1 \ s_1 \ f \ x)\}. & \\
S \neq \{\} \longrightarrow & \\
f = f' (\subseteq vs_1, vs_1', \cup \{tags \ cs \ vs_1 \ s_1 \ f \ x \mid x. x \in S\}) \longrightarrow & \\
(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge & \\
(c_2 = SKIP) = (c_2' = SKIP) \wedge & \\
\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) \ vs_2. \text{fst } p \in S] = & \\
\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') \ vs_2'. \text{fst } p \in S] \wedge & \\
s_2 = t_2 (\subseteq S) &
\end{aligned}$$

**abbreviation *ok-flow-2* where**

$$\begin{aligned}
\text{ok-flow-2 } c_1 \ c_2 \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ cs &\equiv \\
\forall S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } cs \ vs_1 \ s_1 \ f \ x)\}. & \\
S \neq \{\} \longrightarrow & \\
f = f' (\subseteq vs_1, vs_1', \cup \{tags-out \ cs \ vs_1 \ s_1 \ f \ x \mid x. x \in S\}) \longrightarrow & \\
(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge & \\
(c_2 = SKIP) = (c_2' = SKIP) \wedge & \\
\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) \ vs_2. \text{fst } p \in S] = & \\
\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') \ vs_2'. \text{fst } p \in S] \wedge & \\
[p \leftarrow \text{drop } (\text{length } ws_1) \ ws_2. \text{fst } p \in S] = & \\
[p \leftarrow \text{drop } (\text{length } ws_1') \ ws_2'. \text{fst } p \in S] &
\end{aligned}$$

**abbreviation *ok-flow* where**

$$\begin{aligned}
\text{ok-flow } c_1 \ c_2 \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ cs &\equiv \\
\forall t_1 \ f' \ vs_1' \ ws_1'. \exists c_2' \ t_2 \ vs_2' \ ws_2'. & \\
\text{ok-flow-1 } c_1 \ c_2 \ c_2' \ s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ cs \wedge & \\
\text{ok-flow-2 } c_1 \ c_2 \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ cs &
\end{aligned}$$

**definition *correct* :: com  $\Rightarrow$  state set  $\Rightarrow$  vname set  $\Rightarrow$  bool where**

$$\begin{aligned}
\text{correct } c \ A \ X &\equiv \\
\forall s \in \text{Univ } A (\subseteq \text{state} \cap X). \forall c_1 \ c_2 \ s_1 \ s_2 \ f \ vs \ vs_1 \ vs_2 \ ws \ ws_1 \ ws_2 \ cfs. & \\
(c, s, f, vs, ws) \rightarrow^* (c_1, s_1, f, vs_1, ws_1) \wedge & \\
(c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2) \longrightarrow & \\
\text{ok-flow } c_1 \ c_2 \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ (\text{flow } cfs) &
\end{aligned}$$

**abbreviation *noninterf-set* :: state set  $\Rightarrow$  vname set  $\Rightarrow$  vname set  $\Rightarrow$  bool**

$$\begin{aligned}
\langle \langle \cdot \mid \cdot \rangle \rangle [51, 51, 51] \ 50) \text{ where} & \\
A: X \rightsquigarrow \mid Y &\equiv \forall y \in Y. \exists s \in A. \exists x \in X. \neg s: \text{dom } x \rightsquigarrow \text{dom } y
\end{aligned}$$

**abbreviation *ok-flow-aux-1* where**

$$\begin{aligned}
\text{ok-flow-aux-1 } c_1 \ c_2 \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ cs &\equiv \\
\forall S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } cs \ vs_1 \ s_1 \ f \ x)\}. &
\end{aligned}$$

$$\begin{aligned}
& S \neq \{\} \longrightarrow \\
& f = f' (\subseteq vs_1, vs_1', \cup \{tags\text{-aux } cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x \mid x. x \in S\}) \longrightarrow \\
& (c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge \\
& (c_2 = SKIP) = (c_2' = SKIP) \wedge \\
& map \text{ fst } [p \leftarrow drop \text{ (length } vs_1) \text{ } vs_2. \text{ fst } p \in S] = \\
& map \text{ fst } [p \leftarrow drop \text{ (length } vs_1') \text{ } vs_2'. \text{ fst } p \in S]
\end{aligned}$$

**abbreviation** *ok-flow-aux-2* **where**

$$\begin{aligned}
& ok\text{-flow-aux-2 } s_1 \text{ } s_2 \text{ } t_1 \text{ } t_2 \text{ } f \text{ } f' \text{ } vs_1 \text{ } vs_1' \text{ } cs \equiv \\
& \forall S \subseteq \{x. s_1 = t_1 (\subseteq sources \text{ } cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x)\}. \\
& S \neq \{\} \longrightarrow \\
& f = f' (\subseteq vs_1, vs_1', \cup \{tags \text{ } cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x \mid x. x \in S\}) \longrightarrow \\
& s_2 = t_2 (\subseteq S)
\end{aligned}$$

**abbreviation** *ok-flow-aux-3* **where**

$$\begin{aligned}
& ok\text{-flow-aux-3 } s_1 \text{ } t_1 \text{ } f \text{ } f' \text{ } vs_1 \text{ } vs_1' \text{ } ws_1 \text{ } ws_1' \text{ } ws_2 \text{ } ws_2' \text{ } cs \equiv \\
& \forall S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-out } cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x)\}. \\
& S \neq \{\} \longrightarrow \\
& f = f' (\subseteq vs_1, vs_1', \cup \{tags\text{-out } cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x \mid x. x \in S\}) \longrightarrow \\
& [p \leftarrow drop \text{ (length } ws_1) \text{ } ws_2. \text{ fst } p \in S] = \\
& [p \leftarrow drop \text{ (length } ws_1') \text{ } ws_2'. \text{ fst } p \in S]
\end{aligned}$$

**abbreviation** *ok-flow-aux* :: *config set*  $\Rightarrow$  *com*  $\Rightarrow$  *com*  $\Rightarrow$  *state*  $\Rightarrow$  *state*  $\Rightarrow$

*stream*  $\Rightarrow$  *inputs*  $\Rightarrow$  *inputs*  $\Rightarrow$  *outputs*  $\Rightarrow$  *outputs*  $\Rightarrow$  *flow*  $\Rightarrow$  *bool*

**where**

$$\begin{aligned}
& ok\text{-flow-aux } U \text{ } c_1 \text{ } c_2 \text{ } s_1 \text{ } s_2 \text{ } f \text{ } vs_1 \text{ } vs_2 \text{ } ws_1 \text{ } ws_2 \text{ } cs \equiv \\
& (\forall t_1 \text{ } f' \text{ } vs_1' \text{ } ws_1'. \exists c_2' \text{ } t_2 \text{ } vs_2' \text{ } ws_2'. \\
& ok\text{-flow-aux-1 } c_1 \text{ } c_2 \text{ } c_2' \text{ } s_1 \text{ } t_1 \text{ } t_2 \text{ } f \text{ } f' \text{ } vs_1 \text{ } vs_1' \text{ } vs_2 \text{ } vs_2' \text{ } ws_1' \text{ } ws_2' \text{ } cs \wedge \\
& ok\text{-flow-aux-2 } s_1 \text{ } s_2 \text{ } t_1 \text{ } t_2 \text{ } f \text{ } f' \text{ } vs_1 \text{ } vs_1' \text{ } cs \wedge \\
& ok\text{-flow-aux-3 } s_1 \text{ } t_1 \text{ } f \text{ } f' \text{ } vs_1 \text{ } vs_1' \text{ } ws_1 \text{ } ws_1' \text{ } ws_2 \text{ } ws_2' \text{ } cs) \wedge \\
& (\forall Y. (\exists (A, X) \in U. A: X \rightsquigarrow Y) \longrightarrow no\text{-upd } cs \text{ } Y)
\end{aligned}$$

In addition to the equations handling the further constructs of the considered extension of language IMP, the auxiliary recursive function *ctyping1-aux* used to define the idempotent type system *ctyping1* differs from its counterpart used in my previous paper [10] also in that it records any update of a state variable using a pair of type *vname*  $\times$  *val option*, where the first component is the state variable being updated, and the latter one matches *Some i* or *None* depending on whether its new value can be evaluated to an integer *i* at compile time or not.

Apart from the aforesaid type change, the equations for the constructs included in the original language IMP are the same as in my previous paper [10], whereas the equations for the additional constructs of the considered language extension are as follows.

- The equation for an input instruction *IN x*, like the one handling assignments, records the update of variable *x* just in case it is a state

variable (as otherwise its update cannot change the applying interference relation). If so, its update is recorded with  $(x, \text{None})$ , since input values cannot be evaluated at compile time.

- The equation for an output instruction  $OUT\ x$  does not record any update, since output instructions leave the program state unchanged.
- The equation for a nondeterministic choice  $c_1\ OR\ c_2$  sets the returned value to  $\vdash\ c_1\ \sqcup\ \vdash\ c_2$ , in the same way as the equation for a conditional statement  $IF\ b\ THEN\ c_1\ ELSE\ c_2$  whose boolean condition  $b$  cannot be evaluated at compile time.

As in my previous paper [10], the *state set* returned by *ctyping1* is defined so that any *indeterminate* state variable (namely, any state variable  $x$  with a latest recorded update  $(x, \text{None})$ ) may take an arbitrary value. Of course, a real-world implementation of this type system would not need to actually return a distinct state for any such value, but rather just to mark any indeterminate state variable in each returned state with some special value standing for *arbitrary*.

**primrec** *btyping1* :: *bexp*  $\Rightarrow$  *bool option* ( $\langle \vdash - \rangle$  [51] 55) **where**

$\vdash\ Bc\ v = \text{Some } v \mid$

$\vdash\ Not\ b = (\text{case } \vdash\ b\ \text{of}$   
 $\quad \text{Some } v \Rightarrow \text{Some } (\neg v) \mid - \Rightarrow \text{None}) \mid$

$\vdash\ And\ b_1\ b_2 = (\text{case } (\vdash\ b_1, \vdash\ b_2)\ \text{of}$   
 $\quad (\text{Some } v_1, \text{Some } v_2) \Rightarrow \text{Some } (v_1 \wedge v_2) \mid - \Rightarrow \text{None}) \mid$

$\vdash\ Less\ a_1\ a_2 = (\text{if } \text{avars } a_1 \cup \text{avars } a_2 = \{\}$   
 $\quad \text{then } \text{Some } (\text{aval } a_1\ (\lambda x. 0) < \text{aval } a_2\ (\lambda x. 0))\ \text{else } \text{None})$

**inductive-set** *ctyping1-merge-aux* :: *state-upd list set*  $\Rightarrow$   
*state-upd list set*  $\Rightarrow$  (*state-upd list*  $\times$  *bool*) *list set*  
**(infix**  $\langle \sqcup \rangle$  55) **for**  $A$  **and**  $B$  **where**

$xs \in A \Longrightarrow [(xs, \text{True})] \in A \sqcup B \mid$

$ys \in B \Longrightarrow [(ys, \text{False})] \in A \sqcup B \mid$

$\llbracket ws \in A \sqcup B; \neg \text{snd } (\text{hd } ws); xs \in A; (xs, \text{True}) \notin \text{set } ws \rrbracket \Longrightarrow$   
 $(xs, \text{True}) \# ws \in A \sqcup B \mid$

$\llbracket ws \in A \sqcup B; \text{snd } (\text{hd } ws); ys \in B; (ys, \text{False}) \notin \text{set } ws \rrbracket \Longrightarrow$   
 $(ys, \text{False}) \# ws \in A \sqcup B$

**declare** *ctyping1-merge-aux.intros* [intro]

**definition** *ctyping1-append* ::

*state-upd list set*  $\Rightarrow$  *state-upd list set*  $\Rightarrow$  *state-upd list set*  
 (infixl <@> 55) **where**  
 $A @ B \equiv \{xs @ ys \mid xs \text{ ys. } xs \in A \wedge ys \in B\}$

**definition** *ctyping1-merge* ::

*state-upd list set*  $\Rightarrow$  *state-upd list set*  $\Rightarrow$  *state-upd list set*  
 (infixl <⊔> 55) **where**  
 $A \sqcup B \equiv \{\text{concat } (\text{map } \text{fst } ws) \mid ws. ws \in A \sqcup B\}$

**definition** *ctyping1-merge-append* ::

*state-upd list set*  $\Rightarrow$  *state-upd list set*  $\Rightarrow$  *state-upd list set*  
 (infixl <⊔@> 55) **where**  
 $A \sqcup @ B \equiv (\text{if } \text{card } B = \text{Suc } 0 \text{ then } A \text{ else } A \sqcup B) @ B$

**primrec** *ctyping1-aux* :: *com*  $\Rightarrow$  *state-upd list set*

(<⊢ -> [51] 60) **where**

$\vdash \text{SKIP} = \{\{\}\} \mid$

$\vdash x ::= a = (\text{if } x \in \text{state}$   
 then  $\{(x, \text{if } \text{avars } a = \{\} \text{ then } \text{Some } (\text{aval } a \ (\lambda x. 0)) \text{ else } \text{None})\}$   
 else  $\{\{\}\}$ )  $\mid$

$\vdash \text{IN } x = (\text{if } x \in \text{state} \text{ then } \{(x, \text{None})\} \text{ else } \{\{\}\}) \mid$

$\vdash \text{OUT } x = \{\{\}\} \mid$

$\vdash c_1;; c_2 = \vdash c_1 \sqcup @ \vdash c_2 \mid$

$\vdash c_1 \text{ OR } c_2 = \vdash c_1 \sqcup \vdash c_2 \mid$

$\vdash \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 = (\text{let } f = \vdash b \text{ in}$   
 (if  $f \in \{\text{Some True, None}\}$  then  $\vdash c_1$  else  $\{\}$ )  $\sqcup$   
 (if  $f \in \{\text{Some False, None}\}$  then  $\vdash c_2$  else  $\{\}$ ))  $\mid$

$\vdash \text{WHILE } b \text{ DO } c = (\text{let } f = \vdash b \text{ in}$   
 (if  $f \in \{\text{Some False, None}\}$  then  $\{\{\}\}$  else  $\{\}$ )  $\cup$   
 (if  $f \in \{\text{Some True, None}\}$  then  $\vdash c$  else  $\{\}$ ))

**definition** *ctyping1* :: *com*  $\Rightarrow$  *state set*  $\Rightarrow$  *vname set*  $\Rightarrow$  *config*

(<⊢ - '(⊆ -, -)'> [51] 55) **where**

$\vdash c (\subseteq A, X) \equiv \text{let } F = \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in \vdash c\} \text{ in}$   
 ( $\{\lambda x. \text{if } f x = \{\}$

then  $s x$  else case  $\text{snd } (\text{last } (f x))$  of  $\text{None} \Rightarrow t x \mid \text{Some } i \Rightarrow i \mid$

$f \text{ s t. } f \in F \wedge s \in A\},$   
*Univ??*  $A \{x. \forall f \in F. \text{if } f \ x = []$   
*then*  $x \in X \text{ else } \text{snd } (\text{last } (f \ x)) \neq \text{None}\}$

Finally, in the recursive definition of the main type system *ctyping2*, the equations dealing with the constructs included in the original language IMP are the same as in my previous paper [10], whereas the equations for the additional constructs of the considered language extension are as follows.

- The equation for an input instruction *IN*  $x$  sets the returned value to a *pass* verdict *Some*  $(B, Y)$  just in case each set of variables in the current scope is allowed to affect variable  $x$  in the associated set of program states. If so, then the sets  $B$  and  $Y$  are computed in the same way as with an assignment whose right-hand expression cannot be evaluated at compile time, since input values cannot be evaluated at compile time, too.
- The equation for an output instruction *OUT*  $x$  sets the returned value to a *pass* verdict *Some*  $(B, Y)$  just in case each set of variables in the current scope is allowed to affect variable  $x$  in the associated set of program states. If so, then the sets  $B$  and  $Y$  are computed in the same way as with a *SKIP* command, as output instructions leave the program state unchanged, too.
- The equation for a nondeterministic choice  $c_1$  *OR*  $c_2$  sets the returned value to a *pass* verdict *Some*  $(B, Y)$  just in case *pass* verdicts are returned for both branches. If so, then the sets  $B$  and  $Y$  are computed in the same way as with a conditional statement *IF*  $b$  *THEN*  $c_1$  *ELSE*  $c_2$  whose boolean condition  $b$  cannot be evaluated at compile time.

**primrec** *btyping2-aux* :: *bexp*  $\Rightarrow$  *state set*  $\Rightarrow$  *vname set*  $\Rightarrow$  *state set option*  
 ( $\llbracket - \rrbracket (\subseteq -, -')$ ) [51] 55) **where**

$\llbracket Bc \ v \ (\subseteq A, -) \rrbracket = \text{Some } (\text{if } v \text{ then } A \text{ else } \{\}) \mid$

$\llbracket \text{Not } b \ (\subseteq A, X) \rrbracket = (\text{case } \llbracket b \ (\subseteq A, X) \rrbracket \text{ of}$   
*Some*  $B \Rightarrow \text{Some } (A - B) \mid - \Rightarrow \text{None}) \mid$

$\llbracket \text{And } b_1 \ b_2 \ (\subseteq A, X) \rrbracket = (\text{case } (\llbracket b_1 \ (\subseteq A, X) \rrbracket, \llbracket b_2 \ (\subseteq A, X) \rrbracket) \text{ of}$   
*(Some*  $B_1, \text{Some } B_2) \Rightarrow \text{Some } (B_1 \cap B_2) \mid - \Rightarrow \text{None}) \mid$

$\llbracket \text{Less } a_1 \ a_2 \ (\subseteq A, X) \rrbracket = (\text{if } \text{avars } a_1 \cup \text{avars } a_2 \subseteq \text{state} \cap X$   
*then* *Some*  $\{s. s \in A \wedge \text{aval } a_1 \ s < \text{aval } a_2 \ s\}$  *else* *None*)

**definition** *btyping2* ::

$be\!xp \Rightarrow state\ set \Rightarrow vname\ set \Rightarrow state\ set \times state\ set$   
 $(\langle \models - \ '(\subseteq -, -') \rangle [51] 55) \mathbf{where}$   
 $\models b (\subseteq A, X) \equiv case \models b (\subseteq A, X) \text{ of}$   
 $Some\ A' \Rightarrow (A', A - A') \mid - \Rightarrow (A, A)$

**abbreviation**  $interf\text{-}set :: state\ set \Rightarrow vname\ set \Rightarrow vname\ set \Rightarrow bool$   
 $(\langle (-: - \rightsquigarrow -) \rangle [51, 51, 51] 50) \mathbf{where}$   
 $A: X \rightsquigarrow Y \equiv \forall s \in A. \forall x \in X. \forall y \in Y. s: dom\ x \rightsquigarrow dom\ y$

**abbreviation**  $atyping :: bool \Rightarrow aexp \Rightarrow vname\ set \Rightarrow bool$   
 $(\langle (- \models - \ '(\subseteq -')) \rangle [51, 51] 50) \mathbf{where}$   
 $v \models a (\subseteq X) \equiv avars\ a = \{\} \vee avars\ a \subseteq state \cap X \wedge v$

**definition**  $univ\text{-}states\text{-}if :: state\ set \Rightarrow vname\ set \Rightarrow state\ set$   
 $(\langle (Univ? - -) \rangle [51, 75] 75) \mathbf{where}$   
 $Univ? A X \equiv if\ state \subseteq X \text{ then } A \text{ else } Univ\ A (\subseteq \{\})$

**fun**  $ctyping2 :: scope \Rightarrow com \Rightarrow state\ set \Rightarrow vname\ set \Rightarrow config\ option$   
 $(\langle (- \models - \ '(\subseteq -, -')) \rangle [51, 51] 55) \mathbf{where}$

$- \models SKIP (\subseteq A, X) = Some\ (A, Univ??\ A\ X) \mid$

$(U, v) \models x ::= a (\subseteq A, X) =$   
 $(if\ \forall (B, Y) \in insert\ (Univ? A\ X, avars\ a)\ U. B: Y \rightsquigarrow \{x\}$   
 $then\ Some\ (if\ x \in state \wedge A \neq \{\}$   
 $then\ if\ v \models a (\subseteq X)$   
 $then\ (\{s(x := aval\ a\ s) \mid s. s \in A\}, insert\ x\ X) \text{ else } (A, X - \{x\})$   
 $else\ (A, Univ??\ A\ X))$   
 $else\ None) \mid$

$(U, v) \models IN\ x (\subseteq A, X) =$   
 $(if\ \forall (B, Y) \in U. B: Y \rightsquigarrow \{x\}$   
 $then\ Some\ (if\ x \in state \wedge A \neq \{\}$   
 $then\ (A, X - \{x\}) \text{ else } (A, Univ??\ A\ X))$   
 $else\ None) \mid$

$(U, v) \models OUT\ x (\subseteq A, X) =$   
 $(if\ \forall (B, Y) \in U. B: Y \rightsquigarrow \{x\}$   
 $then\ Some\ (A, Univ??\ A\ X)$   
 $else\ None) \mid$

$(U, v) \models c_1;; c_2 (\subseteq A, X) =$   
 $(case\ (U, v) \models c_1 (\subseteq A, X) \text{ of}$   
 $Some\ (B, Y) \Rightarrow (U, v) \models c_2 (\subseteq B, Y) \mid - \Rightarrow None) \mid$

$(U, v) \models c_1\ OR\ c_2 (\subseteq A, X) =$   
 $(case\ ((U, v) \models c_1 (\subseteq A, X), (U, v) \models c_2 (\subseteq A, X)) \text{ of}$

$$\begin{aligned}
& (Some (C_1, Y_1), Some (C_2, Y_2)) \Rightarrow Some (C_1 \cup C_2, Y_1 \cap Y_2) \mid \\
& - \Rightarrow None) \mid \\
(U, v) \models IF b THEN c_1 ELSE c_2 (\subseteq A, X) = \\
& (case (insert (Univ? A X, bvars b) U, \models b (\subseteq A, X)) of (U', B_1, B_2) \Rightarrow \\
& case ((U', v) \models c_1 (\subseteq B_1, X), (U', v) \models c_2 (\subseteq B_2, X)) of \\
& (Some (C_1, Y_1), Some (C_2, Y_2)) \Rightarrow Some (C_1 \cup C_2, Y_1 \cap Y_2) \mid \\
& - \Rightarrow None) \mid \\
(U, v) \models WHILE b DO c (\subseteq A, X) = (case \models b (\subseteq A, X) of (B_1, B_2) \Rightarrow \\
& case \vdash c (\subseteq B_1, X) of (C, Y) \Rightarrow case \models b (\subseteq C, Y) of (B_1', B_2') \Rightarrow \\
& if \forall (B, W) \in insert (Univ? A X \cup Univ? C Y, bvars b) U. B: W \rightsquigarrow UNIV \\
& then case ((\{\}, False) \models c (\subseteq B_1, X), (\{\}, False) \models c (\subseteq B_1', Y)) of \\
& (Some -, Some -) \Rightarrow Some (B_2 \cup B_2', Univ?? B_2 X \cap Y) \mid \\
& - \Rightarrow None \\
& else None) \\
\text{end} \\
\text{end}
\end{aligned}$$

### 3 Idempotence of the auxiliary type system meant for loop bodies

**theory** *Idempotence*  
**imports** *Definitions*  
**begin**

As in my previous paper [10], the purpose of this section is to prove that the auxiliary type system *ctyping1* used to simulate the execution of loop bodies is idempotent, namely that if its output for a given input is the pair formed by *state set*  $B$  and *varname set*  $Y$ , then the output is the same if  $B$  and  $Y$  are fed back into the type system (lemma *ctyping1-idem*).

#### 3.1 Local context proofs

**context** *noninterf*  
**begin**

**abbreviation** *ctyping1-idem-lhs* **where**

*ctyping1-idem-lhs*  $s \ t \ t' \ ys \ ys' \ x \equiv$

*if*  $[y \leftarrow ys'. \text{fst } y = x] = []$

*then if*  $[y \leftarrow ys. \text{fst } y = x] = []$

*then*  $s \ x$

*else case*  $\text{snd } (\text{last } [y \leftarrow ys. \text{fst } y = x])$  *of*  $None \Rightarrow t \ x \mid \text{Some } i \Rightarrow i$

*else case*  $\text{snd } (\text{last } [y \leftarrow ys'. \text{fst } y = x])$  *of*  $None \Rightarrow t' \ x \mid \text{Some } i \Rightarrow i$

**abbreviation** *ctyping1-idem-rhs* **where**

*ctyping1-idem-rhs*  $f\ s\ t\ x \equiv$   
 if  $f\ x = []$   
 then  $s\ x$   
 else case *snd* (*last* ( $f\ x$ )) of *None*  $\Rightarrow t\ x$  | *Some*  $i \Rightarrow i$

**abbreviation** *ctyping1-idem-pred* **where**

*ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (S :: \text{state-upd list set}) \equiv \exists f\ s'.$   
 $(\exists t''. \text{ctyping1-idem-lhs } s\ t\ t'\ ys\ ys'\ = \text{ctyping1-idem-rhs } f\ s'\ t'') \wedge$   
 $(\forall x. (f\ x = [] \iff [y \leftarrow ys @ ys'. \text{fst } y = x] = [])) \wedge$   
 $(f\ x \neq [] \implies \text{last } (f\ x) = \text{last } [y \leftarrow ys @ ys'. \text{fst } y = x]) \wedge$   
 $(\exists ys''. f = (\lambda x. [y \leftarrow ys''. \text{fst } y = x]) \wedge ys'' \in S) \wedge s' \in A$

**lemma** *ctyping1-merge-aux-no-nil*:

$ws \in A \sqcup B \implies ws \neq []$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-merge-aux-empty-lhs*:

$\{\} \sqcup B = \{[(ys, \text{False})] \mid ys. ys \in B\}$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-merge-aux-empty-rhs*:

$A \sqcup \{\} = \{[(xs, \text{True})] \mid xs. xs \in A\}$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-merge-empty-lhs*:

$\{\} \sqcup B = B$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-merge-empty-rhs*:

$A \sqcup \{\} = A$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-aux-nonempty*:

$\vdash c \neq \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-merge-idem-fst*:

**assumes**

$A: \bigwedge ys\ ys'. ys \in \vdash c_1 \implies ys' \in \vdash c_1 \implies$   
 $\text{ctyping1-idem-pred } s\ t\ t'\ ys\ ys'\ A\ (\vdash c_1)$  **and**  
 $B: \bigwedge ys\ ys'. ys \in \vdash c_2 \implies ys' \in \vdash c_2 \implies$   
 $\text{ctyping1-idem-pred } s\ t\ t'\ ys\ ys'\ A\ (\vdash c_2)$  **and**  
 $C: s \in A$  **and**  
 $D: ys \in \vdash c_1 \sqcup \vdash c_2$  **and**  
 $E: ys' \in \vdash c_1 \sqcup \vdash c_2$

**shows** *ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_1\ \sqcup\ \vdash\ c_2)$   
 ⟨*proof*⟩

**lemma** *ctyping1-merge-append-idem-fst*:

**assumes**

$A: \bigwedge ys\ ys'. ys \in \vdash c_1 \implies ys' \in \vdash c_1 \implies$   
*ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_1)$  **and**

$B: \bigwedge ys\ ys'. ys \in \vdash c_2 \implies ys' \in \vdash c_2 \implies$   
*ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_2)$  **and**

$C: s \in A$  **and**

$D: ys \in \vdash c_1 \sqcup_{\text{@}} \vdash c_2$  **and**

$E: ys' \in \vdash c_1 \sqcup_{\text{@}} \vdash c_2$

**shows** *ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_1 \sqcup_{\text{@}} \vdash\ c_2)$   
 ⟨*proof*⟩

**lemma** *ctyping1-aux-idem-fst*:

$\llbracket s \in A; ys \in \vdash c; ys' \in \vdash c \rrbracket \implies$   
*ctyping1-idem-pred*  $s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c)$   
 ⟨*proof*⟩

**lemma** *ctyping1-idem-fst-1*:

$\llbracket s \in A; ys \in \vdash c; ys' \in \vdash c \rrbracket \implies \exists f\ s'.$   
 $(\exists t''. \text{ctyping1-idem-lhs}\ s\ t\ t'\ ys\ ys' = \text{ctyping1-idem-rhs}\ f\ s'\ t'') \wedge$   
 $(\exists ys''. f = (\lambda x. [y \leftarrow ys''. \text{fst}\ y = x]) \wedge ys'' \in \vdash c) \wedge s' \in A$   
 ⟨*proof*⟩

**lemma** *ctyping1-idem-fst-2*:

$\llbracket s \in A; ys \in \vdash c \rrbracket \implies \exists f\ s'.$   
 $(\exists t'.$   
 $(\lambda x. \text{if } [y \leftarrow ys. \text{fst}\ y = x] = []$   
 $\text{then } s\ x$   
 $\text{else case snd (last } [y \leftarrow ys. \text{fst}\ y = x]) \text{ of None } \Rightarrow t\ x \mid \text{Some } i \Rightarrow i) =$   
 $(\lambda x. \text{if } f\ x = []$   
 $\text{then } s'\ x$   
 $\text{else case snd (last (f } x)) \text{ of None } \Rightarrow t'\ x \mid \text{Some } i \Rightarrow i)) \wedge$   
 $(\exists ys'. f = (\lambda x. [y \leftarrow ys'. \text{fst}\ y = x]) \wedge ys' \in \vdash c) \wedge$   
 $(\exists f'\ s''.$   
 $(\exists t''. s' = (\lambda x. \text{if } f'\ x = []$   
 $\text{then } s''\ x$   
 $\text{else case snd (last (f'\ } x)) \text{ of None } \Rightarrow t''\ x \mid \text{Some } i \Rightarrow i)) \wedge$   
 $(\exists ys''. f' = (\lambda x. [y \leftarrow ys''. \text{fst}\ y = x]) \wedge ys'' \in \vdash c) \wedge s'' \in A)$   
 $(\text{is } \llbracket -; - \rrbracket \implies \exists - -. (\exists -. ?f = -) \wedge -)$   
 ⟨*proof*⟩

**lemma** *ctyping1-idem-fst*:

$\vdash c (\subseteq A, X) = (B, Y) \implies \text{case } \vdash c (\subseteq B, Y) \text{ of } (B', Y') \Rightarrow B' = B$   
 ⟨*proof*⟩

**lemma** *ctyping1-idem-snd-1*:

**assumes**

$A: A \neq \{\}$  **and**

$B: \forall r f s.$

$(\forall t. r \neq (\lambda x. \text{if } f x = [] \text{ then } s x \text{ else case snd (last (f x)) of$   
 $\text{None} \Rightarrow t x \mid \text{Some } i \Rightarrow i)) \vee$

$(\forall ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \longrightarrow ys \notin \vdash c) \vee s \notin A$

**(is**  $\forall r f s. (\forall t. r \neq ?r f s t) \vee -)$

**shows**  $UNIV = S$

*<proof>*

**lemma** *ctyping1-idem-snd-2*:

$\{x. \forall f.$

$(f x = [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$

$(\forall f.$

$(f x = [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$   
 $x \in X) \wedge$

$(f x \neq [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$   
 $(\exists i. \text{snd (last (f x)) = Some } i))) \wedge$

$(f x \neq [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$

$(\exists i. \text{snd (last (f x)) = Some } i))\} =$

$\{x. \forall f.$

$(f x = [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$   
 $x \in X) \wedge$

$(f x \neq [] \longrightarrow (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$

$(\exists i. \text{snd (last (f x)) = Some } i))\}$

*<proof>*

**lemma** *ctyping1-idem-snd*:

$\vdash c (\subseteq A, X) = (B, Y) \implies \text{case } \vdash c (\subseteq B, Y) \text{ of } (B', Y') \Rightarrow Y' = Y$

*<proof>*

**lemma** *ctyping1-idem*:

$\vdash c (\subseteq A, X) = (B, Y) \implies \vdash c (\subseteq B, Y) = (B, Y)$

*<proof>*

**end**

**end**

## 4 Overapproximation of program semantics by the main type system

**theory** *Overapproximation*

**imports** *Idempotence*

**begin**

As in my previous paper [10], the purpose of this section is to prove that type system *ctyping2* overapproximates program semantics, namely that if (a)  $(c, s, p) \Rightarrow (t, q)$ , (b) the type system outputs a *state set*  $B$  and a *vname set*  $Y$  when it is input program  $c$ , *state set*  $A$ , and *vname set*  $X$ , and (c) state  $s$  agrees with some state in  $A$  on the value of each state variable in  $X$ , then  $t$  must agree with some state in  $B$  on the value of each state variable in  $Y$  (lemma *ctyping2-approx*).

This proof makes use of the lemma *ctyping1-idem* proven in the previous section.

## 4.1 Global context proofs

**lemma** *avars-aval*:

$s = t (\subseteq \text{avars } a) \implies \text{aval } a \ s = \text{aval } a \ t$   
*<proof>*

## 4.2 Local context proofs

**context** *noninterf*

**begin**

**lemma** *interf-set-mono*:

$\llbracket A' \subseteq A; X \subseteq X'; \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y; \forall (B, Y) \in \text{insert } (\text{Univ? } A \ X, Z) \ U. B: Y \rightsquigarrow W \rrbracket \implies$   
 $\forall (B, Y) \in \text{insert } (\text{Univ? } A' \ X', Z) \ U'. B: Y \rightsquigarrow W$   
*<proof>*

**lemma** *btyping1-btyping2-aux-1* [elim]:

**assumes**

$A: \text{avars } a_1 = \{\}$  **and**

$B: \text{avars } a_2 = \{\}$  **and**

$C: \text{aval } a_1 (\lambda x. 0) < \text{aval } a_2 (\lambda x. 0)$

**shows**  $\text{aval } a_1 \ s < \text{aval } a_2 \ s$

*<proof>*

**lemma** *btyping1-btyping2-aux-2* [elim]:

**assumes**

$A: \text{avars } a_1 = \{\}$  **and**

$B: \text{avars } a_2 = \{\}$  **and**

$C: \neg \text{aval } a_1 (\lambda x. 0) < \text{aval } a_2 (\lambda x. 0)$  **and**

$D: \text{aval } a_1 \ s < \text{aval } a_2 \ s$

**shows** *False*

*<proof>*

**lemma** *btyping1-btyping2-aux*:

$\vdash b = \text{Some } v \implies \models b (\subseteq A, X) = \text{Some (if } v \text{ then } A \text{ else } \{\})$   
 $\langle \text{proof} \rangle$

**lemma** *btyping1-btyping2*:

$\vdash b = \text{Some } v \implies \models b (\subseteq A, X) = (\text{if } v \text{ then } (A, \{\}) \text{ else } (\{\}, A))$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-subset*:

$\models b (\subseteq A, X) = \text{Some } A' \implies A' = \{s. s \in A \wedge \text{bval } b \ s\}$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-diff*:

$\models b (\subseteq A, X) = \text{Some } B; \models b (\subseteq A', X') = \text{Some } B'; A' \subseteq A; B' \subseteq B \implies$   
 $A' - B' \subseteq A - B$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-mono*:

$\models b (\subseteq A, X) = \text{Some } B; A' \subseteq A; X \subseteq X' \implies$   
 $\exists B'. \models b (\subseteq A', X') = \text{Some } B' \wedge B' \subseteq B$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-mono*:

$\models b (\subseteq A, X) = (B_1, B_2); \models b (\subseteq A', X') = (B_1', B_2'); A' \subseteq A; X \subseteq X' \implies$   
 $B_1' \subseteq B_1 \wedge B_2' \subseteq B_2$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-un-eq*:

$\models b (\subseteq A, X) = (B_1, B_2) \implies B_1 \cup B_2 = A$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-eq*:

$\models b (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X) \implies \text{bval } b \ s = \text{bval } b \ t$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-mono-fst*:

$\vdash c (\subseteq A, X) = (B, Y); \vdash c (\subseteq A', X') = (B', Y'); A' \subseteq A \implies$   
 $B' \subseteq B$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping1-mono*:

**assumes**

$A: \vdash c (\subseteq A, X) = (B, Y)$  **and**  
 $B: \vdash c (\subseteq A', X') = (B', Y')$  **and**  
 $C: A' \subseteq A$  **and**  
 $D: X \subseteq X'$

**shows**  $B' \subseteq B \wedge Y \subseteq Y'$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-skip* [elim!]:

$$\begin{aligned} & \llbracket (U, \text{False}) \models \text{SKIP} (\subseteq A, X) = \text{Some} (C, Z); A' \subseteq A; X \subseteq X' \rrbracket \implies \\ & \quad \exists C' Z'. (U', \text{False}) \models \text{SKIP} (\subseteq A', X') = \text{Some} (C', Z') \wedge \\ & \quad C' \subseteq C \wedge Z \subseteq Z' \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-assign* [elim!]:

$$\begin{aligned} & \llbracket (U, \text{False}) \models x ::= a (\subseteq A, X) = \text{Some} (C, Z); A' \subseteq A; X \subseteq X'; \\ & \quad \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies \\ & \quad \exists C' Z'. (U', \text{False}) \models x ::= a (\subseteq A', X') = \text{Some} (C', Z') \wedge \\ & \quad C' \subseteq C \wedge Z \subseteq Z' \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-input* [elim!]:

$$\begin{aligned} & \llbracket (U, \text{False}) \models \text{IN } x (\subseteq A, X) = \text{Some} (C, Z); A' \subseteq A; X \subseteq X'; \\ & \quad \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies \\ & \quad \exists C' Z'. (U', \text{False}) \models \text{IN } x (\subseteq A', X') = \text{Some} (C', Z') \wedge \\ & \quad C' \subseteq C \wedge Z \subseteq Z' \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-output* [elim!]:

$$\begin{aligned} & \llbracket (U, \text{False}) \models \text{OUT } x (\subseteq A, X) = \text{Some} (C, Z); A' \subseteq A; X \subseteq X'; \\ & \quad \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies \\ & \quad \exists C' Z'. (U', \text{False}) \models \text{OUT } x (\subseteq A', X') = \text{Some} (C', Z') \wedge \\ & \quad C' \subseteq C \wedge Z \subseteq Z' \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-seq*:

**assumes**

$$A: \bigwedge A' B X' Y U'.$$

$$(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some} (B, Y) \implies A' \subseteq A \implies X \subseteq X' \implies$$

$$\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$$

$$\exists B' Y'. (U', \text{False}) \models c_1 (\subseteq A', X') = \text{Some} (B', Y') \wedge$$

$$B' \subseteq B \wedge Y \subseteq Y' \text{ and}$$

$$B: \bigwedge p B Y B' C Y' Z U'.$$

$$(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } p \implies (B, Y) = p \implies$$

$$(U, \text{False}) \models c_2 (\subseteq B, Y) = \text{Some} (C, Z) \implies B' \subseteq B \implies Y \subseteq Y' \implies$$

$$\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$$

$$\exists C' Z'. (U', \text{False}) \models c_2 (\subseteq B', Y') = \text{Some} (C', Z') \wedge$$

$$C' \subseteq C \wedge Z \subseteq Z' \text{ and}$$

$$C: (U, \text{False}) \models c_1;; c_2 (\subseteq A, X) = \text{Some} (C, Z) \text{ and}$$

$$D: A' \subseteq A \text{ and}$$

$$E: X \subseteq X' \text{ and}$$

$$F: \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y$$

**shows**  $\exists C' Z'. (U', \text{False}) \models c_1;; c_2 (\subseteq A', X') = \text{Some} (C', Z') \wedge$

$$C' \subseteq C \wedge Z \subseteq Z'$$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-or*:

**assumes**

$A: \bigwedge A' C_1 X' Y_1 U'.$

$(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C_1, Y_1) \implies A' \subseteq A \implies X \subseteq X' \implies$   
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists C_1' Y_1'. (U', \text{False}) \models c_1 (\subseteq A', X') = \text{Some } (C_1', Y_1') \wedge$   
 $C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1' \text{ and}$

$B: \bigwedge A' C_2 X' Y_2 U'.$

$(U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C_2, Y_2) \implies A' \subseteq A \implies X \subseteq X' \implies$   
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists C_2' Y_2'. (U', \text{False}) \models c_2 (\subseteq A', X') = \text{Some } (C_2', Y_2') \wedge$   
 $C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2' \text{ and}$

$C: (U, \text{False}) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C, Y) \text{ and}$

$D: A' \subseteq A \text{ and}$

$E: X \subseteq X' \text{ and}$

$F: \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y$

**shows**  $\exists C' Y'. (U', \text{False}) \models c_1 \text{ OR } c_2 (\subseteq A', X') = \text{Some } (C', Y') \wedge$   
 $C' \subseteq C \wedge Y \subseteq Y'$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-if*:

**assumes**

$A: \bigwedge W p B_1 B_2 B_1' C_1 X' Y_1 W'. (W, p) =$

$(\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies (B_1, B_2) = p \implies$   
 $(W, \text{False}) \models c_1 (\subseteq B_1, X) = \text{Some } (C_1, Y_1) \implies B_1' \subseteq B_1 \implies$   
 $X \subseteq X' \implies \forall (B', Y') \in W'. \exists (B, Y) \in W. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists C_1' Y_1'. (W', \text{False}) \models c_1 (\subseteq B_1', X') = \text{Some } (C_1', Y_1') \wedge$   
 $C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1' \text{ and}$

$B: \bigwedge W p B_1 B_2 B_2' C_2 X' Y_2 W'. (W, p) =$

$(\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies (B_1, B_2) = p \implies$   
 $(W, \text{False}) \models c_2 (\subseteq B_2, X) = \text{Some } (C_2, Y_2) \implies B_2' \subseteq B_2 \implies$   
 $X \subseteq X' \implies \forall (B', Y') \in W'. \exists (B, Y) \in W. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists C_2' Y_2'. (W', \text{False}) \models c_2 (\subseteq B_2', X') = \text{Some } (C_2', Y_2') \wedge$   
 $C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2' \text{ and}$

$C: (U, \text{False}) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (C, Y) \text{ and}$

$D: A' \subseteq A \text{ and}$

$E: X \subseteq X' \text{ and}$

$F: \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y$

**shows**  $\exists C' Y'. (U', \text{False}) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A', X') =$   
 $\text{Some } (C', Y') \wedge C' \subseteq C \wedge Y \subseteq Y'$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-mono-while*:

**assumes**

$A: \bigwedge B_1 B_2 C Y B_1' B_2' D_1 E X' V U'. (B_1, B_2) = \models b (\subseteq A, X) \implies$

$(C, Y) = \vdash c (\subseteq B_1, X) \implies (B_1', B_2') = \models b (\subseteq C, Y) \implies$

$\forall (B, W) \in \text{insert } (\text{Univ? } A X \cup \text{Univ? } C Y, \text{bvars } b) U.$

$B: W \rightsquigarrow UNIV \implies$   
 $(\{\}, False) \models c (\subseteq B_1, X) = Some (E, V) \implies D_1 \subseteq B_1 \implies$   
 $X \subseteq X' \implies \forall (B', Y') \in U'. \exists (B, Y) \in \{\}. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists E' V'. (U', False) \models c (\subseteq D_1, X') = Some (E', V') \wedge$   
 $E' \subseteq E \wedge V \subseteq V' \text{ and}$   
 $B: \bigwedge B_1 B_2 C Y B_1' B_2' D_1' F Y' W U'. (B_1, B_2) = \models b (\subseteq A, X) \implies$   
 $(C, Y) = \vdash c (\subseteq B_1, X) \implies (B_1', B_2') = \models b (\subseteq C, Y) \implies$   
 $\forall (B, W) \in insert (Univ? A X \cup Univ? C Y, bvars b) U.$   
 $B: W \rightsquigarrow UNIV \implies$   
 $(\{\}, False) \models c (\subseteq B_1', Y) = Some (F, W) \implies D_1' \subseteq B_1' \implies$   
 $Y \subseteq Y' \implies \forall (B', Y') \in U'. \exists (B, Y) \in \{\}. B' \subseteq B \wedge Y' \subseteq Y \implies$   
 $\exists F' W'. (U', False) \models c (\subseteq D_1', Y') = Some (F', W') \wedge$   
 $F' \subseteq F \wedge W \subseteq W' \text{ and}$   
 $C: (U, False) \models WHILE b DO c (\subseteq A, X) = Some (B, Z) \text{ and}$   
 $D: A' \subseteq A \text{ and}$   
 $E: X \subseteq X' \text{ and}$   
 $F: \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y$   
**shows**  $\exists B' Z'. (U', False) \models WHILE b DO c (\subseteq A', X') = Some (B', Z') \wedge$   
 $B' \subseteq B \wedge Z \subseteq Z'$   
 $\langle proof \rangle$

**lemma** *ctyping2-mono*:

$\llbracket (U, False) \models c (\subseteq A, X) = Some (C, Z); A' \subseteq A; X \subseteq X';$   
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies$   
 $\exists C' Z'. (U', False) \models c (\subseteq A', X') = Some (C', Z') \wedge C' \subseteq C \wedge Z \subseteq Z'$   
 $\langle proof \rangle$

**lemma** *ctyping1-ctyping2-fst-assign* [elim!]:

**assumes**  
 $A: \vdash x ::= a (\subseteq A, X) = (C, Z) \text{ and}$   
 $B: (U, False) \models x ::= a (\subseteq A, X) = Some (C', Z')$   
**shows**  $C' \subseteq C$   
 $\langle proof \rangle$

**lemma** *ctyping1-ctyping2-fst-input* [elim!]:

**assumes**  
 $A: \vdash IN x (\subseteq A, X) = (C, Z) \text{ and}$   
 $B: (U, False) \models IN x (\subseteq A, X) = Some (C', Z')$   
**shows**  $C' \subseteq C$   
 $\langle proof \rangle$

**lemma** *ctyping1-ctyping2-fst-output* [elim!]:

$\llbracket \vdash OUT x (\subseteq A, X) = (C, Z);$   
 $(U, False) \models OUT x (\subseteq A, X) = Some (C', Z') \rrbracket \implies$   
 $C' \subseteq C$   
 $\langle proof \rangle$

**lemma** *ctyping1-ctyping2-fst-seq*:

**assumes**

$A: \vdash c_1;; c_2 (\subseteq A, X) = (C, Z)$  **and**  
 $B: (U, \text{False}) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (C', Z')$  **and**  
 $C: \bigwedge B B' Y Y'. \vdash c_1 (\subseteq A, X) = (B, Y) \implies$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (B', Y') \implies B' \subseteq B$  **and**  
 $D: \bigwedge p B' Y' D' C' W' Z'.$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } p \implies (B', Y') = p \implies$   
 $\vdash c_2 (\subseteq B', Y') = (D', W') \implies$   
 $(U, \text{False}) \models c_2 (\subseteq B', Y') = \text{Some } (C', Z') \implies C' \subseteq D'$

**shows**  $C' \subseteq C$

*<proof>*

**lemma** *ctyping1-ctyping2-fst-or*:

**assumes**

$A: \vdash c_1$  **OR**  $c_2 (\subseteq A, X) = (C, Y)$  **and**  
 $B: (U, \text{False}) \models c_1$  **OR**  $c_2 (\subseteq A, X) = \text{Some } (C', Y')$  **and**  
 $C: \bigwedge C C' Y Y'. \vdash c_1 (\subseteq A, X) = (C, Y) \implies$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C', Y') \implies C' \subseteq C$  **and**  
 $D: \bigwedge C C' Y Y'. \vdash c_2 (\subseteq A, X) = (C, Y) \implies$   
 $(U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C', Y') \implies C' \subseteq C$

**shows**  $C' \subseteq C$

*<proof>*

**lemma** *ctyping1-ctyping2-fst-if*:

**assumes**

$A: \vdash$  **IF**  $b$  **THEN**  $c_1$  **ELSE**  $c_2 (\subseteq A, X) = (C, Y)$  **and**  
 $B: (U, \text{False}) \models$  **IF**  $b$  **THEN**  $c_1$  **ELSE**  $c_2 (\subseteq A, X) = \text{Some } (C', Y')$  **and**  
 $C: \bigwedge U' p B_1 B_2 C C' Y Y'.$   
 $(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$   
 $(B_1, B_2) = p \implies \vdash c_1 (\subseteq B_1, X) = (C, Y) \implies$   
 $(U', \text{False}) \models c_1 (\subseteq B_1, X) = \text{Some } (C', Y') \implies C' \subseteq C$  **and**  
 $D: \bigwedge U' p B_1 B_2 C C' Y Y'.$   
 $(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$   
 $(B_1, B_2) = p \implies \vdash c_2 (\subseteq B_2, X) = (C, Y) \implies$   
 $(U', \text{False}) \models c_2 (\subseteq B_2, X) = \text{Some } (C', Y') \implies C' \subseteq C$

**shows**  $C' \subseteq C$

*<proof>*

**lemma** *ctyping1-ctyping2-fst-while*:

**assumes**

$A: \vdash$  **WHILE**  $b$  **DO**  $c (\subseteq A, X) = (B, Z)$  **and**  
 $B: (U, \text{False}) \models$  **WHILE**  $b$  **DO**  $c (\subseteq A, X) = \text{Some } (B', Z')$

**shows**  $B' \subseteq B$

*<proof>*

**lemma** *ctyping1-ctyping2-fst*:

$\llbracket \vdash c (\subseteq A, X) = (C, Z); (U, \text{False}) \models c (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $C' \subseteq C$

*<proof>*

**lemma** *ctyping1-ctyping2-snd-skip* [elim!]:

$\llbracket \vdash \text{SKIP } (\subseteq A, X) = (C, Z);$   
 $(U, \text{False}) \models \text{SKIP } (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $Z \subseteq Z'$   
 ⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-assign* [elim!]:

$\llbracket \vdash x ::= a (\subseteq A, X) = (C, Z);$   
 $(U, \text{False}) \models x ::= a (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $Z \subseteq Z'$   
 ⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-input* [elim!]:

$\llbracket \vdash \text{IN } x (\subseteq A, X) = (C, Z);$   
 $(U, \text{False}) \models \text{IN } x (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $Z \subseteq Z'$   
 ⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-output* [elim!]:

$\llbracket \vdash \text{OUT } x (\subseteq A, X) = (C, Z);$   
 $(U, \text{False}) \models \text{OUT } x (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $Z \subseteq Z'$   
 ⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-seq*:

**assumes**

$A: \vdash c_1;; c_2 (\subseteq A, X) = (C, Z)$  **and**  
 $B: (U, \text{False}) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (C', Z')$  **and**  
 $C: \bigwedge B B' Y Y'. \vdash c_1 (\subseteq A, X) = (B, Y) \implies$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (B', Y') \implies Y \subseteq Y'$  **and**  
 $D: \bigwedge p B' Y' D' C' W' Z'.$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } p \implies (B', Y') = p \implies$   
 $\vdash c_2 (\subseteq B', Y') = (D', W') \implies$   
 $(U, \text{False}) \models c_2 (\subseteq B', Y') = \text{Some } (C', Z') \implies W' \subseteq Z'$

**shows**  $Z \subseteq Z'$

⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-or*:

**assumes**

$A: \vdash c_1 \text{ OR } c_2 (\subseteq A, X) = (C, Y)$  **and**  
 $B: (U, \text{False}) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C', Y')$  **and**  
 $C: \bigwedge C C' Y Y'. \vdash c_1 (\subseteq A, X) = (C, Y) \implies$   
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C', Y') \implies Y \subseteq Y'$  **and**  
 $D: \bigwedge C C' Y Y'. \vdash c_2 (\subseteq A, X) = (C, Y) \implies$   
 $(U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C', Y') \implies Y \subseteq Y'$

**shows**  $Y \subseteq Y'$

⟨proof⟩

**lemma** *ctyping1-ctyping2-snd-if*:

**assumes**

$A: \vdash \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = (C, Y) \text{ and}$

$B: (U, \text{False}) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (C', Y') \text{ and}$

$C: \bigwedge U' p B_1 B_2 C C' Y Y'.$

$(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$

$(B_1, B_2) = p \implies \vdash c_1 (\subseteq B_1, X) = (C, Y) \implies$

$(U', \text{False}) \models c_1 (\subseteq B_1, X) = \text{Some } (C', Y') \implies Y \subseteq Y' \text{ and}$

$D: \bigwedge U' p B_1 B_2 C C' Y Y'.$

$(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$

$(B_1, B_2) = p \implies \vdash c_2 (\subseteq B_2, X) = (C, Y) \implies$

$(U', \text{False}) \models c_2 (\subseteq B_2, X) = \text{Some } (C', Y') \implies Y \subseteq Y'$

**shows**  $Y \subseteq Y'$

*<proof>*

**lemma** *ctyping1-ctyping2-snd-while*:

**assumes**

$A: \vdash \text{WHILE } b \text{ DO } c (\subseteq A, X) = (B, Z) \text{ and}$

$B: (U, \text{False}) \models \text{WHILE } b \text{ DO } c (\subseteq A, X) = \text{Some } (B', Z')$

**shows**  $Z \subseteq Z'$

*<proof>*

**lemma** *ctyping1-ctyping2-snd*:

$\llbracket \vdash c (\subseteq A, X) = (C, Z); (U, \text{False}) \models c (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $Z \subseteq Z'$

*<proof>*

**lemma** *ctyping1-ctyping2*:

$\llbracket \vdash c (\subseteq A, X) = (C, Z); (U, \text{False}) \models c (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \implies$   
 $C' \subseteq C \wedge Z \subseteq Z'$

*<proof>*

**lemma** *btyping2-aux-approx-1* [elim]:

**assumes**

$A: \models b_1 (\subseteq A, X) = \text{Some } B_1 \text{ and}$

$B: \models b_2 (\subseteq A, X) = \text{Some } B_2 \text{ and}$

$C: \text{bval } b_1 \text{ s and}$

$D: \text{bval } b_2 \text{ s and}$

$E: r \in A \text{ and}$

$F: s = r (\subseteq \text{state} \cap X)$

**shows**  $\exists r' \in B_1 \cap B_2. r = r' (\subseteq \text{state} \cap X)$

*<proof>*

**lemma** *btyping2-aux-approx-2* [elim]:

**assumes**

$A: \text{avars } a_1 \subseteq \text{state} \text{ and}$

$B: \text{avars } a_2 \subseteq \text{state} \text{ and}$   
 $C: \text{avars } a_1 \subseteq X \text{ and}$   
 $D: \text{avars } a_2 \subseteq X \text{ and}$   
 $E: \text{aval } a_1 \ s < \text{aval } a_2 \ s \text{ and}$   
 $F: r \in A \text{ and}$   
 $G: s = r (\subseteq \text{state} \cap X)$   
**shows**  $\exists r'. r' \in A \wedge \text{aval } a_1 \ r' < \text{aval } a_2 \ r' \wedge r = r' (\subseteq \text{state} \cap X)$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-approx-3* [elim]:

**assumes**  
 $A: \text{avars } a_1 \subseteq \text{state} \text{ and}$   
 $B: \text{avars } a_2 \subseteq \text{state} \text{ and}$   
 $C: \text{avars } a_1 \subseteq X \text{ and}$   
 $D: \text{avars } a_2 \subseteq X \text{ and}$   
 $E: \neg \text{aval } a_1 \ s < \text{aval } a_2 \ s \text{ and}$   
 $F: r \in A \text{ and}$   
 $G: s = r (\subseteq \text{state} \cap X)$   
**shows**  $\exists r' \in A - \{s \in A. \text{aval } a_1 \ s < \text{aval } a_2 \ s\}. r = r' (\subseteq \text{state} \cap X)$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-aux-approx*:

$\llbracket \models b (\subseteq A, X) = \text{Some } A'; s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$   
 $s \in \text{Univ} \text{ (if } \text{bval } b \ s \text{ then } A' \text{ else } A - A') (\subseteq \text{state} \cap X)$   
 $\langle \text{proof} \rangle$

**lemma** *btyping2-approx*:

$\llbracket \models b (\subseteq A, X) = (B_1, B_2); s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$   
 $s \in \text{Univ} \text{ (if } \text{bval } b \ s \text{ then } B_1 \text{ else } B_2) (\subseteq \text{state} \cap X)$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping2-approx-assign* [elim!]:

$\llbracket \forall t'. \text{aval } a \ s = t' \ x \longrightarrow (\forall s. t' = s(x := \text{aval } a \ s) \longrightarrow s \notin A) \vee$   
 $(\exists y \in \text{state} \cap X. y \neq x \wedge t \ y \neq t' \ y);$   
 $v \models a (\subseteq X); t \in A; s = t (\subseteq \text{state} \cap X) \rrbracket \implies \text{False}$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping2-approx-if-1*:

$\llbracket \text{bval } b \ s; \models b (\subseteq A, X) = (B_1, B_2); r \in A; s = r (\subseteq \text{state} \cap X);$   
 $(\text{insert } (\text{Univ? } A \ X, \text{bvars } b) \ U, v) \models c_1 (\subseteq B_1, X) = \text{Some } (C_1, Y_1);$   
 $\bigwedge A \ B \ X \ Y \ U \ v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$   
 $\exists r \in A. s = r (\subseteq \text{state} \cap X) \implies \exists r' \in B. t = r' (\subseteq \text{state} \cap Y) \rrbracket \implies$   
 $\exists r' \in C_1 \cup C_2. t = r' (\subseteq \text{state} \cap (Y_1 \cap Y_2))$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping2-approx-if-2*:

$\llbracket \neg \text{bval } b \ s; \models b (\subseteq A, X) = (B_1, B_2); r \in A; s = r (\subseteq \text{state} \cap X);$   
 $(\text{insert } (\text{Univ? } A \ X, \text{bvars } b) \ U, v) \models c_2 (\subseteq B_2, X) = \text{Some } (C_2, Y_2);$

$\bigwedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies$   
 $\exists r \in A. s = r (\subseteq \text{state} \cap X) \implies \exists r' \in B. t = r' (\subseteq \text{state} \cap Y) \implies$   
 $\exists r' \in C_1 \cup C_2. t = r' (\subseteq \text{state} \cap (Y_1 \cap Y_2))$   
 <proof>

**lemma** *ctyping2-approx-while-1* [elim]:

$\llbracket \neg \text{bval } b \ s; r \in A; s = r (\subseteq \text{state} \cap X); \models b (\subseteq A, X) = (B, \{\}) \rrbracket \implies$   
 $\exists t \in C. s = t (\subseteq \text{state} \cap Y)$   
 <proof>

**lemma** *ctyping2-approx-while-2* [elim]:

$\llbracket \forall t \in B_2 \cup B_2'. \exists x \in \text{state} \cap (X \cap Y). r \ x \neq t \ x; \neg \text{bval } b \ s;$   
 $r \in A; s = r (\subseteq \text{state} \cap X); \models b (\subseteq A, X) = (B_1, B_2) \rrbracket \implies \text{False}$   
 <proof>

**lemma** *ctyping2-approx-while-aux*:

assumes

A:  $\models b (\subseteq A, X) = (B_1, B_2)$  and

B:  $\vdash c (\subseteq B_1, X) = (C, Y)$  and

C:  $\models b (\subseteq C, Y) = (B_1', B_2')$  and

D:  $(\{\}, \text{False}) \models c (\subseteq B_1, X) = \text{Some } (D, Z)$  and

E:  $(\{\}, \text{False}) \models c (\subseteq B_1', Y) = \text{Some } (D', Z')$  and

F:  $r_1 \in A$  and

G:  $s_1 = r_1 (\subseteq \text{state} \cap X)$  and

H:  $\text{bval } b \ s_1$  and

I:  $\bigwedge C B Y W U. (\text{case } \models b (\subseteq C, Y) \text{ of } (B_1', B_2') \Rightarrow$

$\text{case } \vdash c (\subseteq B_1', Y) \text{ of } (C', Y') \Rightarrow$

$\text{case } \models b (\subseteq C', Y') \text{ of } (B_1'', B_2'') \Rightarrow \text{if}$

$(\forall s \in \text{Univ? } C \ Y \cup \text{Univ? } C' \ Y'. \forall x \in \text{bvars } b. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y) \wedge$

$(\forall p \in U. \text{case } p \text{ of } (B, W) \Rightarrow \forall s \in B. \forall x \in W. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y)$

$\text{then case } (\{\}, \text{False}) \models c (\subseteq B_1', Y) \text{ of}$

$\text{None} \Rightarrow \text{None} \mid \text{Some } - \Rightarrow \text{case } (\{\}, \text{False}) \models c (\subseteq B_1'', Y') \text{ of}$

$\text{None} \Rightarrow \text{None} \mid \text{Some } - \Rightarrow \text{Some } (B_2' \cup B_2'', \text{Univ?? } B_2' \ Y \cap Y')$

$\text{else None}) = \text{Some } (B, W) \implies$

$\exists r \in C. s_2 = r (\subseteq \text{state} \cap Y) \implies \exists r \in B. s_3 = r (\subseteq \text{state} \cap W)$

(is  $\bigwedge C B Y W U. ?P \ C \ B \ Y \ W \ U \implies - \implies -$ ) and

J:  $\bigwedge A B X Y U v. (U, v) \models c (\subseteq A, X) = \text{Some } (B, Y) \implies$

$\exists r \in A. s_1 = r (\subseteq \text{state} \cap X) \implies \exists r \in B. s_2 = r (\subseteq \text{state} \cap Y)$  and

K:  $\forall s \in \text{Univ? } A \ X \cup \text{Univ? } C \ Y. \forall x \in \text{bvars } b. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y$  and

L:  $\forall p \in U. \forall B \ W. p = (B, W) \longrightarrow$

$(\forall s \in B. \forall x \in W. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y)$

shows  $\exists r \in B_2 \cup B_2'. s_3 = r (\subseteq \text{state} \cap \text{Univ?? } B_2 \ X \cap Y)$

<proof>

**lemmas** *ctyping2-approx-while-3* =

*ctyping2-approx-while-aux* [where  $B_2 = \{\}$ , simplified]

**lemma** *ctyping2-approx-while-4*:

$\llbracket \models b (\subseteq A, X) = (B_1, B_2);$

```

 $\vdash c (\subseteq B_1, X) = (C, Y);$ 
 $\models b (\subseteq C, Y) = (B_1', B_2');$ 
 $(\{\}, False) \models c (\subseteq B_1, X) = Some (D, Z);$ 
 $(\{\}, False) \models c (\subseteq B_1', Y) = Some (D', Z');$ 
 $r_1 \in A; s_1 = r_1 (\subseteq state \cap X); bval\ b\ s_1;$ 
 $\bigwedge C\ B\ Y\ W\ U. (case\ \models b (\subseteq C, Y)\ of\ (B_1', B_2') \Rightarrow$ 
   $case\ \vdash c (\subseteq B_1', Y)\ of\ (C', Y') \Rightarrow$ 
   $case\ \models b (\subseteq C', Y')\ of\ (B_1'', B_2'') \Rightarrow$ 
   $if\ (\forall s \in Univ? C\ Y \cup Univ? C'\ Y'. \forall x \in bvars\ b. \forall y. s: dom\ x \rightsquigarrow dom\ y) \wedge$ 
   $(\forall p \in U. case\ p\ of\ (B, W) \Rightarrow \forall s \in B. \forall x \in W. \forall y. s: dom\ x \rightsquigarrow dom\ y)$ 
   $then\ case\ (\{\}, False) \models c (\subseteq B_1', Y)\ of$ 
   $None \Rightarrow None \mid Some\ - \Rightarrow case\ (\{\}, False) \models c (\subseteq B_1'', Y')\ of$ 
   $None \Rightarrow None \mid Some\ - \Rightarrow Some\ (B_2' \cup B_2'', Univ?? B_2'\ Y \cap Y')$ 
   $else\ None) = Some\ (B, W) \Rightarrow$ 
   $\exists r \in C. s_2 = r (\subseteq state \cap Y) \Rightarrow \exists r \in B. s_3 = r (\subseteq state \cap W);$ 
 $\bigwedge A\ B\ X\ Y\ U\ v. (U, v) \models c (\subseteq A, X) = Some\ (B, Y) \Rightarrow$ 
   $\exists r \in A. s_1 = r (\subseteq state \cap X) \Rightarrow \exists r \in B. s_2 = r (\subseteq state \cap Y);$ 
 $\forall s \in Univ? A\ X \cup Univ? C\ Y. \forall x \in bvars\ b. \forall y. s: dom\ x \rightsquigarrow dom\ y;$ 
 $\forall p \in U. \forall B\ W. p = (B, W) \longrightarrow (\forall s \in B. \forall x \in W. \forall y. s: dom\ x \rightsquigarrow dom\ y);$ 
 $\forall r \in B_2 \cup B_2'. \exists x \in state \cap (X \cap Y). s_3\ x \neq r\ x] \Rightarrow$ 
   $False$ 
 $\langle proof \rangle$ 

```

**lemma** *ctyping2-approx*:

```

 $\llbracket (c, s, p) \Rightarrow (t, q); (U, v) \models c (\subseteq A, X) = Some (B, Y);$ 
 $s \in Univ\ A (\subseteq state \cap X) \rrbracket \Rightarrow t \in Univ\ B (\subseteq state \cap Y)$ 
 $\langle proof \rangle$ 

```

**end**

**end**

## 5 Sufficiency of well-typedness for information flow correctness: propaedeutic lemmas

**theory** *Correctness-Lemmas*

**imports** *Overapproximation*

**begin**

The purpose of this section is to prove some further lemmas used in the proof of the main theorem, which is the subject of the next section.

The proof of one of these lemmas uses the lemmas *ctyping1-idem* and *ctyping2-approx* proven in the previous sections.

### 5.1 Global context proofs

**lemma** *bvars-bval*:

$s = t (\subseteq \text{bvars } b) \implies \text{bval } b \ s = \text{bval } b \ t$   
 ⟨proof⟩

**lemma** *eq-streams-subset*:

$\llbracket f = f' (\subseteq \text{vs}, \text{vs}', T); T' \subseteq T \rrbracket \implies f = f' (\subseteq \text{vs}, \text{vs}', T')$   
 ⟨proof⟩

**lemma** *flow-append-1*:

**assumes**  $A: \bigwedge \text{cfs}' :: (\text{com} \times \text{stage}) \text{ list.}$

$c \# \text{map fst } (\text{cfs} :: (\text{com} \times \text{stage}) \text{ list}) = \text{map fst } \text{cfs}' \implies$

$\text{flow-aux } (\text{map fst } \text{cfs}' @ \text{map fst } \text{cfs}'') =$

$\text{flow-aux } (\text{map fst } \text{cfs}') @ \text{flow-aux } (\text{map fst } \text{cfs}'')$

**shows**  $\text{flow-aux } (c \# \text{map fst } \text{cfs} @ \text{map fst } \text{cfs}'') =$

$\text{flow-aux } (c \# \text{map fst } \text{cfs}) @ \text{flow-aux } (\text{map fst } \text{cfs}'')$

⟨proof⟩

**lemma** *flow-append*:

$\text{flow } (\text{cfs} @ \text{cfs}') = \text{flow } \text{cfs} @ \text{flow } \text{cfs}'$   
 ⟨proof⟩

**lemma** *flow-cons*:

$\text{flow } (\text{cf} \# \text{cfs}) = \text{flow-aux } (\text{fst } \text{cf} \# []) @ \text{flow } \text{cfs}$   
 ⟨proof⟩

**lemma** *in-flow-length*:

$\text{length } [p \leftarrow \text{in-flow } \text{cs } \text{vs } f. \text{fst } p = x] = \text{length } [c \leftarrow \text{cs}. c = \text{IN } x]$   
 ⟨proof⟩

**lemma** *in-flow-append*:

$\text{in-flow } (\text{cs} @ \text{cs}') \text{vs } f =$   
 $\text{in-flow } \text{cs } \text{vs } f @ \text{in-flow } \text{cs}' (\text{vs} @ \text{in-flow } \text{cs } \text{vs } f) f$   
 ⟨proof⟩

**lemma** *in-flow-one*:

$\text{in-flow } [c] \text{vs } f = (\text{case } c \text{ of}$   
 $\text{IN } x \Rightarrow [(x, f \ x (\text{length } [p \leftarrow \text{vs}. \text{fst } p = x]))] \mid - \Rightarrow [])$   
 ⟨proof⟩

**lemma** *run-flow-append*:

$\text{run-flow } (\text{cs} @ \text{cs}') \text{vs } s \ f =$   
 $\text{run-flow } \text{cs}' (\text{vs} @ \text{in-flow } \text{cs } \text{vs } f) (\text{run-flow } \text{cs } \text{vs } s \ f) f$   
 ⟨proof⟩

**lemma** *run-flow-one*:

$\text{run-flow } [c] \text{vs } s \ f = (\text{case } c \text{ of}$   
 $x ::= a \Rightarrow s(x := \text{aval } a \ s) \mid$

$IN\ x \Rightarrow s(x := f\ x\ (\text{length}\ [p \leftarrow vs.\ \text{fst}\ p = x])) \mid$   
 $\quad - \Rightarrow s)$   
 <proof>

**lemma** *run-flow-observe*:  
 $run\text{-}flow\ (\langle X \rangle \# cs)\ vs\ s\ f = run\text{-}flow\ cs\ vs\ s\ f$   
 <proof>

**lemma** *out-flow-append*:  
 $out\text{-}flow\ (cs\ @\ cs')\ vs\ s\ f =$   
 $\quad out\text{-}flow\ cs\ vs\ s\ f\ @$   
 $\quad out\text{-}flow\ cs'\ (vs\ @\ in\text{-}flow\ cs\ vs\ f)\ (run\text{-}flow\ cs\ vs\ s\ f)\ f$   
 <proof>

**lemma** *out-flow-one*:  
 $out\text{-}flow\ [c]\ vs\ s\ f = (\text{case}\ c\ \text{of}$   
 $\quad OUT\ x \Rightarrow [(x, s\ x)] \mid - \Rightarrow [])$   
 <proof>

**lemma** *no-upd-empty*:  
 $no\text{-}upd\ cs\ \{\}$   
 <proof>

**lemma** *no-upd-append*:  
 $no\text{-}upd\ (cs\ @\ cs')\ X = (no\text{-}upd\ cs\ X \wedge no\text{-}upd\ cs'\ X)$   
 <proof>

**lemma** *no-upd-in-flow*:  
 $no\text{-}upd\ cs\ X \Longrightarrow [p \leftarrow in\text{-}flow\ cs\ vs\ f.\ \text{fst}\ p \in X] = []$   
 <proof>

**lemma** *no-upd-run-flow*:  
 $no\text{-}upd\ cs\ X \Longrightarrow run\text{-}flow\ cs\ vs\ s\ f = s\ (\subseteq\ X)$   
 <proof>

**lemma** *no-upd-out-flow*:  
 $no\text{-}upd\ cs\ X \Longrightarrow [p \leftarrow out\text{-}flow\ cs\ vs\ s\ f.\ \text{fst}\ p \in X] = []$   
 <proof>

**lemma** *small-stepsl-append*:  
 $\llbracket cf \rightarrow^*\{cfs\}\ cf';\ cf' \rightarrow^*\{cfs'\}\ cf'' \rrbracket \Longrightarrow cf \rightarrow^*\{cfs\ @\ cfs'\}\ cf''$   
 <proof>

**lemma** *small-step-stream*:  
 $(c, s, f, vs, ws) \rightarrow (c', p) \Longrightarrow \exists s'\ vs'\ ws'.\ p = (s', f, vs', ws')$   
 <proof>

**lemma** *small-stepsl-stream*:

$$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', p) \implies \exists s' vs' ws'. p = (s', f, vs', ws')$$

*<proof>*

**lemma** *small-steps-steps1*:

$$\exists cfs. cf \rightarrow^*\{cfs\} cf$$

*<proof>*

**lemma** *small-steps-steps2*:

$$\llbracket cf \rightarrow cf'; cf' \rightarrow^*\{cfs\} cf'' \rrbracket \implies \exists cfs'. cf \rightarrow^*\{cfs'\} cf''$$

*<proof>*

**lemma** *small-steps-steps1*:

$$cf \rightarrow^* cf' \implies \exists cfs. cf \rightarrow^*\{cfs\} cf'$$

*<proof>*

**lemma** *small-stepsl-steps*:

$$cf \rightarrow^*\{cfs\} cf' \implies cf \rightarrow^* cf'$$

*<proof>*

**lemma** *small-steps-stream*:

$$(c, s, f, vs, ws) \rightarrow^* (c', p) \implies \exists s' vs' ws'. p = (s', f, vs', ws')$$

*<proof>*

**lemma** *small-stepsl-cons-1*:

$$cf \rightarrow^*\{[cf']\} cf'' \implies cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{\} cf'')$$

*<proof>*

**lemma** *small-stepsl-cons-2*:

$$\begin{aligned} \llbracket cf \rightarrow^*\{cf' \# cfs\} cf'' \rrbracket &\implies \\ cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs\} cf''); & \\ cf \rightarrow^*\{cf' \# cfs @ [cf'']\} cf''' \rrbracket &\implies \\ cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs @ [cf'']\} cf''') & \end{aligned}$$

*<proof>*

**lemma** *small-stepsl-cons*:

$$\begin{aligned} cf \rightarrow^*\{cf' \# cfs\} cf'' &\implies \\ cf' = cf \wedge & \\ (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs\} cf'') & \end{aligned}$$

*<proof>*

**lemma** *small-stepsl-skip*:

$$(SKIP, p) \rightarrow^*\{cfs\} cf \implies cf = (SKIP, p) \wedge flow\ cfs = \square$$

*<proof>*

**lemma** *small-stepsl-assign*:

$$\begin{aligned}
& (x ::= a, s, p) \rightarrow^*\{cfs\} cf \implies \\
& \quad cf = (x ::= a, s, p) \wedge \\
& \quad \text{flow } cfs = [] \vee \\
& \quad cf = (SKIP, s(x := \text{aval } a \ s), p) \wedge \\
& \quad \text{flow } cfs = [x ::= a] \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *small-stepsl-input*:

$$\begin{aligned}
& (IN \ x, s, f, vs, ws) \rightarrow^*\{cfs\} cf \implies \\
& \quad cf = (IN \ x, s, f, vs, ws) \wedge \\
& \quad \text{flow } cfs = [] \vee \\
& \quad (\text{let } n = \text{length } [p \leftarrow vs. \text{fst } p = x] \\
& \quad \text{in } cf = (SKIP, s(x := f \ x \ n), f, vs @ [(x, f \ x \ n)], ws) \wedge \\
& \quad \text{flow } cfs = [IN \ x]) \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *small-stepsl-output*:

$$\begin{aligned}
& (OUT \ x, s, f, vs, ws) \rightarrow^*\{cfs\} cf \implies \\
& \quad cf = (OUT \ x, s, f, vs, ws) \wedge \\
& \quad \text{flow } cfs = [] \vee \\
& \quad cf = (SKIP, s, f, vs, ws @ [(x, s \ x)]) \wedge \\
& \quad \text{flow } cfs = [OUT \ x] \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *small-stepsl-seq-1*:

$$\begin{aligned}
& (c_1;; c_2, p) \rightarrow^*\{[]\} (c, q) \implies \\
& \quad (\exists c' \ cfs'. c = c';; c_2 \wedge \\
& \quad \quad (c_1, p) \rightarrow^*\{cfs'\} (c', q) \wedge \\
& \quad \quad \text{flow } [] = \text{flow } cfs') \vee \\
& \quad (\exists p' \ cfs' \ cfs''. \text{length } cfs'' < \text{length } [] \wedge \\
& \quad \quad (c_1, p) \rightarrow^*\{cfs'\} (SKIP, p') \wedge \\
& \quad \quad (c_2, p') \rightarrow^*\{cfs''\} (c, q) \wedge \\
& \quad \quad \text{flow } [] = \text{flow } cfs' @ \text{flow } cfs'') \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *small-stepsl-seq-2*:

$$\begin{aligned}
& \text{assumes } A: \bigwedge c' \ q'. cf = (c', q') \implies \\
& \quad (c_1;; c_2, p) \rightarrow^*\{cfs\} (c', q') \implies \\
& \quad \quad (\exists c'' \ cfs'. c' = c'';; c_2 \wedge \\
& \quad \quad \quad (c_1, p) \rightarrow^*\{cfs'\} (c'', q') \wedge \\
& \quad \quad \quad \text{flow } cfs = \text{flow } cfs') \vee \\
& \quad \quad (\exists p' \ cfs' \ cfs''. \text{length } cfs'' < \text{length } cfs \wedge \\
& \quad \quad \quad (c_1, p) \rightarrow^*\{cfs'\} (SKIP, p') \wedge \\
& \quad \quad \quad (c_2, p') \rightarrow^*\{cfs''\} (c', q') \wedge \\
& \quad \quad \quad \text{flow } cfs = \text{flow } cfs' @ \text{flow } cfs'') \\
& \quad (\text{is } \bigwedge c' \ q'. - \implies - \implies \\
& \quad \quad (\exists c'' \ cfs'. ?P \ c' \ q' \ c'' \ cfs') \vee
\end{aligned}$$

$(\exists p' cfs' cfs''. ?Q c' q' p' cfs' cfs'')$   
**assumes**  $B: (c_1;; c_2, p) \rightarrow*\{cfs @ [cf]\} (c, q)$   
**shows**  
 $(\exists c' cfs'. c = c';; c_2 \wedge$   
 $(c_1, p) \rightarrow*\{cfs'\} (c', q) \wedge$   
 $flow (cfs @ [cf]) = flow cfs') \vee$   
 $(\exists p' cfs' cfs''. length cfs'' < length (cfs @ [cf]) \wedge$   
 $(c_1, p) \rightarrow*\{cfs'\} (SKIP, p') \wedge$   
 $(c_2, p') \rightarrow*\{cfs''\} (c, q) \wedge$   
 $flow (cfs @ [cf]) = flow cfs' @ flow cfs'')$   
**(is**  $?T \vee ?U)$   
 $\langle proof \rangle$

**lemma** *small-stepsl-seq*:

$(c_1;; c_2, p) \rightarrow*\{cfs\} (c, q) \implies$   
 $(\exists c' cfs'. c = c';; c_2 \wedge$   
 $(c_1, p) \rightarrow*\{cfs'\} (c', q) \wedge$   
 $flow cfs = flow cfs') \vee$   
 $(\exists p' cfs' cfs''. length cfs'' < length cfs \wedge$   
 $(c_1, p) \rightarrow*\{cfs'\} (SKIP, p') \wedge (c_2, p') \rightarrow*\{cfs''\} (c, q) \wedge$   
 $flow cfs = flow cfs' @ flow cfs'')$   
 $\langle proof \rangle$

**lemma** *small-stepsl-or-1*:

**assumes**  $A: (c_1 OR c_2, p) \rightarrow*\{cfs\} cf \implies$   
 $cf = (c_1 OR c_2, p) \wedge$   
 $flow cfs = [] \vee$   
 $(c_1, p) \rightarrow*\{tl cfs\} cf \wedge$   
 $flow cfs = flow (tl cfs) \vee$   
 $(c_2, p) \rightarrow*\{tl cfs\} cf \wedge$   
 $flow cfs = flow (tl cfs)$   
**(is**  $- \implies ?P \vee ?Q \vee ?R)$   
**assumes**  $B: (c_1 OR c_2, p) \rightarrow*\{cfs @ [cf]\} cf'$   
**shows**  
 $cf' = (c_1 OR c_2, p) \wedge$   
 $flow (cfs @ [cf]) = [] \vee$   
 $(c_1, p) \rightarrow*\{tl (cfs @ [cf])\} cf' \wedge$   
 $flow (cfs @ [cf]) = flow (tl (cfs @ [cf])) \vee$   
 $(c_2, p) \rightarrow*\{tl (cfs @ [cf])\} cf' \wedge$   
 $flow (cfs @ [cf]) = flow (tl (cfs @ [cf]))$   
**(is**  $- \vee ?T)$   
 $\langle proof \rangle$

**lemma** *small-stepsl-or*:

$(c_1 OR c_2, p) \rightarrow*\{cfs\} cf \implies$   
 $cf = (c_1 OR c_2, p) \wedge$   
 $flow cfs = [] \vee$   
 $(c_1, p) \rightarrow*\{tl cfs\} cf \wedge$

$$\begin{aligned} & \text{flow } cfs = \text{flow } (tl \ cfs) \vee \\ & (c_2, p) \rightarrow^*\{tl \ cfs\} \ cf \wedge \\ & \text{flow } cfs = \text{flow } (tl \ cfs) \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-if-1*:

**assumes** *A*:  $(IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \rightarrow^*\{cfs\} \ cf \implies$   
 $cf = (IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \wedge$   
 $\text{flow } cfs = [] \vee$   
 $bval \ b \ s \wedge (c_1, \ s, \ p) \rightarrow^*\{tl \ cfs\} \ cf \wedge$   
 $\text{flow } cfs = \langle bvars \ b \rangle \# \text{flow } (tl \ cfs) \vee$   
 $\neg \ bval \ b \ s \wedge (c_2, \ s, \ p) \rightarrow^*\{tl \ cfs\} \ cf \wedge$   
 $\text{flow } cfs = \langle bvars \ b \rangle \# \text{flow } (tl \ cfs)$   
**(is**  $- \implies ?P \vee ?Q \vee ?R)$

**assumes** *B*:  $(IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \rightarrow^*\{cfs \ @ \ [cf]\} \ cf'$   
**shows**  
 $cf' = (IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \wedge$   
 $\text{flow } (cfs \ @ \ [cf]) = [] \vee$   
 $bval \ b \ s \wedge (c_1, \ s, \ p) \rightarrow^*\{tl \ (cfs \ @ \ [cf])\} \ cf' \wedge$   
 $\text{flow } (cfs \ @ \ [cf]) = \langle bvars \ b \rangle \# \text{flow } (tl \ (cfs \ @ \ [cf])) \vee$   
 $\neg \ bval \ b \ s \wedge (c_2, \ s, \ p) \rightarrow^*\{tl \ (cfs \ @ \ [cf])\} \ cf' \wedge$   
 $\text{flow } (cfs \ @ \ [cf]) = \langle bvars \ b \rangle \# \text{flow } (tl \ (cfs \ @ \ [cf]))$   
**(is**  $- \vee ?T)$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-if*:

$(IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \rightarrow^*\{cfs\} \ cf \implies$   
 $cf = (IF \ b \ THEN \ c_1 \ ELSE \ c_2, \ s, \ p) \wedge$   
 $\text{flow } cfs = [] \vee$   
 $bval \ b \ s \wedge (c_1, \ s, \ p) \rightarrow^*\{tl \ cfs\} \ cf \wedge$   
 $\text{flow } cfs = \langle bvars \ b \rangle \# \text{flow } (tl \ cfs) \vee$   
 $\neg \ bval \ b \ s \wedge (c_2, \ s, \ p) \rightarrow^*\{tl \ cfs\} \ cf \wedge$   
 $\text{flow } cfs = \langle bvars \ b \rangle \# \text{flow } (tl \ cfs)$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-while-1*:

**assumes** *A*:  $(WHILE \ b \ DO \ c, \ s, \ p) \rightarrow^*\{cfs\} \ cf \implies$   
 $cf = (WHILE \ b \ DO \ c, \ s, \ p) \wedge$   
 $\text{flow } cfs = [] \vee$   
 $bval \ b \ s \wedge (c; \ WHILE \ b \ DO \ c, \ s, \ p) \rightarrow^*\{tl \ cfs\} \ cf \wedge$   
 $\text{flow } cfs = \langle bvars \ b \rangle \# \text{flow } (tl \ cfs) \vee$   
 $\neg \ bval \ b \ s \wedge cf = (SKIP, \ s, \ p) \wedge$   
 $\text{flow } cfs = [\langle bvars \ b \rangle]$   
**(is**  $- \implies ?P \vee ?Q \vee ?R)$

**assumes** *B*:  $(WHILE \ b \ DO \ c, \ s, \ p) \rightarrow^*\{cfs \ @ \ [cf]\} \ cf'$

**shows**

$cf' = (WHILE \ b \ DO \ c, \ s, \ p) \wedge$

$$\begin{aligned}
& \text{flow } (cfs @ [cf]) = [] \vee \\
& \text{bval } b \ s \wedge (c;; \text{WHILE } b \ \text{DO } c, \ s, \ p) \rightarrow^* \{tl \ (cfs @ [cf])\} \ cf' \wedge \\
& \text{flow } (cfs @ [cf]) = \langle \text{bvars } b \rangle \# \text{flow } (tl \ (cfs @ [cf])) \vee \\
& \neg \text{bval } b \ s \wedge cf' = (\text{SKIP}, \ s, \ p) \wedge \\
& \text{flow } (cfs @ [cf]) = [\langle \text{bvars } b \rangle] \\
& (\text{is} - \vee ?T)
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-while*:

$$\begin{aligned}
& (\text{WHILE } b \ \text{DO } c, \ s, \ p) \rightarrow^* \{cfs\} \ cf \implies \\
& \text{cf} = (\text{WHILE } b \ \text{DO } c, \ s, \ p) \wedge \\
& \text{flow } cfs = [] \vee \\
& \text{bval } b \ s \wedge (c;; \text{WHILE } b \ \text{DO } c, \ s, \ p) \rightarrow^* \{tl \ cfs\} \ cf \wedge \\
& \text{flow } cfs = \langle \text{bvars } b \rangle \# \text{flow } (tl \ cfs) \vee \\
& \neg \text{bval } b \ s \wedge cf = (\text{SKIP}, \ s, \ p) \wedge \\
& \text{flow } cfs = [\langle \text{bvars } b \rangle]
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-steps-in-flow-1*:

$$\begin{aligned}
& \llbracket (c, \ s, \ f, \ vs, \ ws) \rightarrow (c', \ s', \ f', \ vs', \ ws') \rrbracket \\
& \text{vs}'' = \text{vs}' @ \text{drop} (\text{length } \text{vs}') \ \text{vs}'' \implies \\
& \text{vs}'' = \text{vs} @ \text{drop} (\text{length } \text{vs}) \ \text{vs}''
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-steps-in-flow*:

$$\begin{aligned}
& (c, \ s, \ f, \ vs, \ ws) \rightarrow^* (c', \ s', \ f', \ vs', \ ws') \implies \\
& \text{vs}' = \text{vs} @ \text{drop} (\text{length } \text{vs}) \ \text{vs}'
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-steps-out-flow-1*:

$$\begin{aligned}
& \llbracket (c, \ s, \ f, \ vs, \ ws) \rightarrow (c', \ s', \ f', \ vs', \ ws') \rrbracket \\
& \text{ws}'' = \text{ws}' @ \text{drop} (\text{length } \text{ws}') \ \text{ws}'' \implies \\
& \text{ws}'' = \text{ws} @ \text{drop} (\text{length } \text{ws}) \ \text{ws}''
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-steps-out-flow*:

$$\begin{aligned}
& (c, \ s, \ f, \ vs, \ ws) \rightarrow^* (c', \ s', \ f', \ vs', \ ws') \implies \\
& \text{ws}' = \text{ws} @ \text{drop} (\text{length } \text{ws}) \ \text{ws}'
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-in-flow-1*:

**assumes**

A:  $(c, \ s, \ f, \ vs, \ ws) \rightarrow^* \{cfs\} (c', \ s', \ f', \ vs @ \text{vs}', \ ws')$  **and**

B:  $(c', \ s', \ f', \ vs @ \text{vs}', \ ws') \rightarrow (c'', \ s'', \ f'', \ \text{vs}'', \ \text{ws}'')$

**shows**  $\text{vs}'' = \text{vs} @ \text{vs}' @$

*in-flow*  $(\text{flow } [(c', \ s', \ f', \ vs @ \text{vs}', \ ws')]) (\text{vs} @ \text{vs}') \ f$

$\langle \text{proof} \rangle$

**lemma** *small-stepsl-in-flow*:

$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', s', f', vs', ws') \implies$   
 $vs' = vs \text{ @ } \text{in-flow} (\text{flow } cfs) vs f$   
 $\langle \text{proof} \rangle$

**lemma** *small-stepsl-run-flow-1*:

**assumes**

A:  $(c, s, f, vs, ws) \rightarrow^*\{cfs\}$   
 $(c', \text{run-flow} (\text{flow } cfs) vs s f, f', vs', ws')$  **and**  
B:  $(c', \text{run-flow} (\text{flow } cfs) vs s f, f', vs', ws') \rightarrow$   
 $(c'', s'', f'', vs'', ws'')$

**shows**  $s'' = \text{run-flow} (\text{flow} [(c', \text{run-flow} (\text{flow } cfs) vs s f, f', vs', ws')])$   
 $(vs \text{ @ } \text{in-flow} (\text{flow } cfs) vs f) (\text{run-flow} (\text{flow } cfs) vs s f) f$   
 $\langle \text{proof} \rangle$

**lemma** *small-stepsl-run-flow*:

$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', s', f', vs', ws') \implies$   
 $s' = \text{run-flow} (\text{flow } cfs) vs s f$   
 $\langle \text{proof} \rangle$

**lemma** *small-stepsl-out-flow-1*:

**assumes**

A:  $(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', s', f', vs', ws \text{ @ } ws')$  **and**  
B:  $(c', s', f', vs', ws \text{ @ } ws') \rightarrow (c'', s'', f'', vs'', ws'')$

**shows**  $ws'' = ws \text{ @ } ws' \text{ @ }$   
 $\text{out-flow} (\text{flow} [(c', s', f', vs', ws \text{ @ } ws')]) (vs \text{ @ } \text{in-flow} (\text{flow } cfs) vs f)$   
 $(\text{run-flow} (\text{flow } cfs) vs s f) f$   
 $\langle \text{proof} \rangle$

**lemma** *small-stepsl-out-flow*:

$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', s', f', vs', ws') \implies$   
 $ws' = ws \text{ @ } \text{out-flow} (\text{flow } cfs) vs s f$   
 $\langle \text{proof} \rangle$

**lemma** *small-steps-inputs*:

**assumes**

A:  $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_0, s_1, f, vs_1, ws_1)$  **and**  
B:  $(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
C:  $(c, s', f', vs_0', ws_0') \rightarrow^* (c_0', s_1', f', vs_1', ws_1')$  **and**  
D:  $(c_1', s_1', f', vs_1', ws_1') \rightarrow^* (c_2', s_2', f', vs_2', ws_2')$  **and**  
E:  $\text{map fst } [p \leftarrow \text{drop} (\text{length } vs_0) vs_1. P p] =$   
 $\text{map fst } [p \leftarrow \text{drop} (\text{length } vs_0') vs_1'. P p]$  **and**  
F:  $\text{map fst } [p \leftarrow \text{drop} (\text{length } vs_1) vs_2. P p] =$   
 $\text{map fst } [p \leftarrow \text{drop} (\text{length } vs_1') vs_2'. P p]$

**shows**  $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0) vs_2. P p] =$   
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0') vs_2'. P p]$   
 ⟨proof⟩

**lemma** *small-steps-outputs*:

**assumes**

$A: (c, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c_0, s_1, f, vs_1, ws_1)$  **and**  
 $B: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
 $C: (c, s', f', vs_0', ws_0') \rightarrow^* (c_0', s_1', f', vs_1', ws_1')$  **and**  
 $D: (c_1', s_1', f', vs_1', ws_1') \rightarrow^* (c_2', s_2', f', vs_2', ws_2')$  **and**  
 $E: [p \leftarrow \text{drop } (\text{length } ws_0) ws_1. P p] =$   
 $[p \leftarrow \text{drop } (\text{length } ws_0') ws_1'. P p]$  **and**  
 $F: [p \leftarrow \text{drop } (\text{length } ws_1) ws_2. P p] =$   
 $[p \leftarrow \text{drop } (\text{length } ws_1') ws_2'. P p]$

**shows**  $[p \leftarrow \text{drop } (\text{length } ws_0) ws_2. P p] =$   
 $[p \leftarrow \text{drop } (\text{length } ws_0') ws_2'. P p]$

⟨proof⟩

## 5.2 Local context proofs

**context** *noninterf*

**begin**

**lemma** *no-upd-sources*:

$\text{no-upd } cs X \implies \forall x \in X. x \in \text{sources } cs \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-aux-append*:

$\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (cs @ cs') \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-out-append*:

$\text{sources-out } cs \text{ vs } s f x \subseteq \text{sources-out } (cs @ cs') \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-aux-sources*:

$\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources } cs \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-aux-sources-out*:

$\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-out } cs \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-aux-observe-hd-1*:

$\forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \implies X \subseteq \text{sources-aux } [(X)] \text{ vs } s f x$   
 ⟨proof⟩

**lemma** *sources-aux-observe-hd-2*:

$\llbracket \forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \implies X \subseteq \text{sources-aux } (\langle X \rangle \# xs) \text{ vs } s f x;$   
 $\forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \rrbracket \implies$   
 $X \subseteq \text{sources-aux } (\langle X \rangle \# xs @ [x']) \text{ vs } s f x$   
 <proof>

**lemma** *sources-aux-observe-hd*:

$\forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \implies X \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$   
 <proof>

**lemma** *sources-aux-bval*:

**assumes**

$A: S \subseteq \{x. s = t (\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# cs) \text{ vs } s f x)\}$  **and**

$B: s \in \text{Univ } A (\subseteq \text{state} \cap X)$  **and**

$C: \text{bval } b s \neq \text{bval } b t$

**shows**  $\text{Univ}^? A X: \text{bvars } b \rightsquigarrow | S$

<proof>

**lemma** *ok-flow-aux-degen*:

**assumes**  $A: \nexists S. S \neq \{\} \wedge S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } cs \text{ vs } s_1 s_1 f x)\}$

**shows**  $\forall c_2' t_2 \text{ vs}_2' \text{ ws}_2'$ .

*ok-flow-aux-1*  $c_1 c_2 c_2' s_1 t_1 t_2 f f' \text{ vs}_1 \text{ vs}_1' \text{ vs}_2 \text{ vs}_2' \text{ ws}_1' \text{ ws}_2' cs \wedge$

*ok-flow-aux-2*  $s_1 s_2 t_1 t_2 f f' \text{ vs}_1 \text{ vs}_1' cs \wedge$

*ok-flow-aux-3*  $s_1 t_1 f f' \text{ vs}_1 \text{ vs}_1' \text{ ws}_1 \text{ ws}_1' \text{ ws}_2 \text{ ws}_2' cs$

(**is**  $\forall c_2' t_2 \text{ vs}_2' \text{ ws}_2'. ?P1 c_2' t_2 \text{ vs}_2' \text{ ws}_2' \wedge ?P2 t_2 \wedge ?P3 \text{ ws}_2'$ )

<proof>

**lemma** *tags-aux-append*:

$\text{tags-aux } cs \text{ vs } s f x \subseteq \text{tags-aux } (cs @ cs') \text{ vs } s f x$   
 <proof>

**lemma** *tags-out-append*:

$\text{tags-out } cs \text{ vs } s f x \subseteq \text{tags-out } (cs @ cs') \text{ vs } s f x$   
 <proof>

**lemma** *tags-aux-tags*:

$\text{tags-aux } cs \text{ vs } s f x \subseteq \text{tags } cs \text{ vs } s f x$   
 <proof>

**lemma** *tags-aux-tags-out*:

$\text{tags-aux } cs \text{ vs } s f x \subseteq \text{tags-out } cs \text{ vs } s f x$   
 <proof>

**lemma** *tags-ubound-1*:

**assumes**

$A: (y, \text{Suc } (\text{length } [c \leftarrow cs. c = \text{IN } y] + n)) \in \text{tags-aux } cs \text{ vs } s f x$  **and**

$B: \bigwedge z n. y = z \implies$

$(z, \text{length } [c \leftarrow cs. c = \text{IN } z] + n) \notin \text{tags-aux } cs \text{ vs } s f x$

**shows** *False*  
*<proof>*

**lemma** *tags-ubound-2*:

**assumes**

*A*:  $(y, \text{Suc} (\text{length} [c \leftarrow \text{cs}. c = \text{IN } y] + n)) \in \text{tags } \text{cs } \text{vs } s f x$  **and**

*B*:  $\bigwedge z n. y = z \implies z \neq x \implies$

$(z, \text{length} [c \leftarrow \text{cs}. c = \text{IN } z] + n) \notin \text{tags } \text{cs } \text{vs } s f x$  **and**

*C*:  $y \neq x$

**shows** *False*

*<proof>*

**lemma** *tags-ubound*:

$(y, \text{length} [c \leftarrow \text{cs}. c = \text{IN } y] + n) \notin \text{tags } \text{cs } \text{vs } s f x$

**and** *tags-aux-ubound*:

$(y, \text{length} [c \leftarrow \text{cs}. c = \text{IN } y] + n) \notin \text{tags-aux } \text{cs } \text{vs } s f x$

*<proof>*

**lemma** *tags-out-ubound-1*:

**assumes**

*A*:  $(y, \text{Suc} (\text{length} [c \leftarrow \text{cs}. c = \text{IN } y] + n)) \in \text{tags-out } \text{cs } \text{vs } s f x$  **and**

*B*:  $\bigwedge z n. y = z \implies$

$(z, \text{length} [c \leftarrow \text{cs}. c = \text{IN } z] + n) \notin \text{tags-out } \text{cs } \text{vs } s f x$

**shows** *False*

*<proof>*

**lemma** *tags-out-ubound*:

$(y, \text{length} [c \leftarrow \text{cs}. c = \text{IN } y] + n) \notin \text{tags-out } \text{cs } \text{vs } s f x$

*<proof>*

**lemma** *tags-less*:

$(y, n) \in \text{tags } \text{cs } \text{vs } s f x \implies n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y]$

*<proof>*

**lemma** *tags-aux-less*:

$(y, n) \in \text{tags-aux } \text{cs } \text{vs } s f x \implies n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y]$

*<proof>*

**lemma** *tags-out-less*:

$(y, n) \in \text{tags-out } \text{cs } \text{vs } s f x \implies n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y]$

*<proof>*

**lemma** *sources-observe-tl-1*:

**assumes**

*A*:  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z = x \implies$

$\text{sources-aux } \text{cs } \text{vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# \text{cs}) \text{ vs } s f x$  **and**

**B:**  $\bigwedge z a b w. c = (z ::= a :: \text{com-flow}) \implies z = x \implies$   
 $\text{sources } cs \text{ vs } s f w \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f w$  **and**  
**C:**  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z \neq x \implies$   
 $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**D:**  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies z = x \implies$   
 $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**E:**  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies z \neq x \implies$   
 $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**F:**  $\bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**G:**  $\bigwedge Y b w. c = \langle Y \rangle \implies$   
 $\text{sources } cs \text{ vs } s f w \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f w$   
**shows**  $\text{sources } (cs @ [c]) \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs @ [c]) \text{ vs } s f x$   
**(is -  $\subseteq$  ?F c)**  
*<proof>*

**lemma** *sources-observe-tl-2:*

**assumes**

**A:**  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**B:**  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies$   
 $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**C:**  $\bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**D:**  $\bigwedge Y. c = \langle Y \rangle \implies$   
 $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**E:**  $\bigwedge Y b w. c = \langle Y \rangle \implies$   
 $\text{sources } cs \text{ vs } s f w \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f w$

**shows**  $\text{sources-aux } (cs @ [c]) \text{ vs } s f x \subseteq$   
 $\text{sources-aux } (\langle X \rangle \# cs @ [c]) \text{ vs } s f x$   
**(is -  $\subseteq$  ?F c)**

*<proof>*

**lemma** *sources-observe-tl:*

$\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$

**and** *sources-aux-observe-tl:*

$\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$   
*<proof>*

**lemma** *sources-out-observe-tl-1:*

**assumes**

**A:**  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $\text{sources-out } cs \text{ vs } s f x \subseteq \text{sources-out } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**B:**  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies$   
 $\text{sources-out } cs \text{ vs } s f x \subseteq \text{sources-out } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**C:**  $\bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $\text{sources-out } cs \text{ vs } s f x \subseteq \text{sources-out } (\langle X \rangle \# cs) \text{ vs } s f x$  **and**  
**D:**  $\bigwedge Y. c = \langle Y \rangle \implies$

$sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$   
**shows**  $sources\text{-}out\ (cs\ @\ [c])\ vs\ s\ f\ x \subseteq$   
 $sources\text{-}out\ (\langle X \rangle \# cs\ @\ [c])\ vs\ s\ f\ x$   
 (is -  $\subseteq$  ?F c)  
 <proof>

**lemma** *sources-out-observe-tl*:  
 $sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$   
 <proof>

**lemma** *tags-observe-tl-1*:  
 $\llbracket \bigwedge z\ a.\ c = z ::= a \implies z = x \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z\ a\ b\ w.\ c = z ::= a \implies z = x \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ w = tags\ cs\ vs\ s\ f\ w;$   
 $\bigwedge z\ a.\ c = z ::= a \implies z \neq x \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z.\ c = IN\ z \implies z = x \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z.\ c = IN\ z \implies z \neq x \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z.\ c = OUT\ z \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$   
 $\bigwedge Y\ b\ w.\ c = \langle Y \rangle \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ w = tags\ cs\ vs\ s\ f\ w \rrbracket \implies$   
 $tags\ (\langle X \rangle \# cs\ @\ [c])\ vs\ s\ f\ x = tags\ (cs\ @\ [c])\ vs\ s\ f\ x$   
 <proof>

**lemma** *tags-observe-tl-2*:  
 $\llbracket \bigwedge z\ a.\ c = z ::= a \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z.\ c = IN\ z \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge z.\ c = OUT\ z \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge Y.\ c = \langle Y \rangle \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x;$   
 $\bigwedge Y\ b\ w.\ c = \langle Y \rangle \implies$   
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ w = tags\ cs\ vs\ s\ f\ w \rrbracket \implies$   
 $tags\text{-}aux\ (\langle X \rangle \# cs\ @\ [c])\ vs\ s\ f\ x = tags\text{-}aux\ (cs\ @\ [c])\ vs\ s\ f\ x$   
 <proof>

**lemma** *tags-observe-tl*:  
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x$   
**and** *tags-aux-observe-tl*:  
 $tags\text{-}aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\text{-}aux\ cs\ vs\ s\ f\ x$   
 <proof>

**lemma** *tags-out-observe-tl-1*:

$$\begin{aligned} & \llbracket \bigwedge z a. c = z ::= a \implies \\ & \quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x; \\ & \bigwedge z. c = IN z \implies \\ & \quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x; \\ & \bigwedge z. c = OUT z \implies \\ & \quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x; \\ & \bigwedge Y. c = \langle Y \rangle \implies \\ & \quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x \rrbracket \implies \\ & \text{tags-out } (\langle X \rangle \# cs @ [c]) \text{ vs } s f x = \text{tags-out } (cs @ [c]) \text{ vs } s f x \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *tags-out-observe-tl*:

$$\text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x$$

*<proof>*

**lemma** *tags-sources-1*:

**assumes**

$$\begin{aligned} A: & \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z = x \implies \\ & (y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies \\ & \quad \text{let } m = \text{Suc } (\text{Max } \{k. k \leq \text{length } cs \wedge \\ & \quad \quad \text{length } [c \leftarrow \text{take } k \text{ cs}. c = IN y] \leq n\}) \\ & \quad \text{in } y \in \text{sources-aux } (\text{drop } m \text{ cs}) \text{ (vs @ in-flow } (\text{take } m \text{ cs}) \text{ vs } f) \\ & \quad \quad (\text{run-flow } (\text{take } m \text{ cs}) \text{ vs } s f) f x \\ & (\text{is } \bigwedge - . - \implies - \implies - \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad - \in \text{sources-aux } - (?G \text{ m cs}) (?H \text{ m cs}) -) \end{aligned}$$

**assumes**

$$\begin{aligned} B: & \bigwedge z a b w. c = (z ::= a :: \text{com-flow}) \implies z = x \implies \\ & (y, n) \in \text{tags } cs \text{ vs } s f w \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f w \textbf{ and} \\ C: & \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z \neq x \implies \\ & (y, n) \in \text{tags } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f x \textbf{ and} \\ D: & \bigwedge z. c = (IN z :: \text{com-flow}) \implies z = x \implies \\ & (y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources-aux } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f x \textbf{ and} \\ E: & \bigwedge z. c = (IN z :: \text{com-flow}) \implies z \neq x \implies \\ & (y, n) \in \text{tags } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f x \textbf{ and} \\ F: & \bigwedge z. c = (OUT z :: \text{com-flow}) \implies \\ & (y, n) \in \text{tags } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f x \textbf{ and} \\ G: & \bigwedge X b w. c = \langle X \rangle \implies \\ & (y, n) \in \text{tags } cs \text{ vs } s f w \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in} \\ & \quad y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) f w \textbf{ and} \\ H: & (y, n) \in \text{tags } (cs @ [c]) \text{ vs } s f x \end{aligned}$$

**shows**  $\text{let } m = \text{Suc } (\text{Max } (?F (cs @ [c]))) \text{ in}$   
 $y \in \text{sources } (\text{drop } m (cs @ [c])) (?G \text{ m } (cs @ [c])) (?H \text{ m } (cs @ [c])) f x$

*<proof>*

**lemma** *tags-sources-2*:

**assumes**

$A: \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies$   
 $\text{let } m = \text{Suc } (\text{Max } \{k. k \leq \text{length } cs \wedge$   
 $\text{length } [c \leftarrow \text{take } k \text{ cs. } c = \text{IN } y] \leq n\})$   
 $\text{in } y \in \text{sources-aux } (\text{drop } m \text{ cs}) \text{ (vs @ in-flow (take } m \text{ cs) vs } f)$   
 $(\text{run-flow (take } m \text{ cs) vs } s f) f x$   
 $(\text{is } \bigwedge - . - \implies - \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in}$   
 $- \in \text{sources-aux } - (?G \text{ } m \text{ cs}) (?H \text{ } m \text{ cs}) - )$

**assumes**

$B: \bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies$   
 $(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in}$   
 $y \in \text{sources-aux } (\text{drop } m \text{ cs}) (?G \text{ } m \text{ cs}) (?H \text{ } m \text{ cs}) f x \text{ and}$   
 $C: \bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in}$   
 $y \in \text{sources-aux } (\text{drop } m \text{ cs}) (?G \text{ } m \text{ cs}) (?H \text{ } m \text{ cs}) f x \text{ and}$   
 $D: \bigwedge X. c = \langle X \rangle \implies$   
 $(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in}$   
 $y \in \text{sources-aux } (\text{drop } m \text{ cs}) (?G \text{ } m \text{ cs}) (?H \text{ } m \text{ cs}) f x \text{ and}$   
 $E: \bigwedge X b w. c = \langle X \rangle \implies$   
 $(y, n) \in \text{tags } cs \text{ vs } s f w \implies \text{let } m = \text{Suc } (\text{Max } (?F \text{ cs})) \text{ in}$   
 $y \in \text{sources } (\text{drop } m \text{ cs}) (?G \text{ } m \text{ cs}) (?H \text{ } m \text{ cs}) f w \text{ and}$   
 $F: (y, n) \in \text{tags-aux } (cs @ [c]) \text{ vs } s f x$   
**shows**  $\text{let } m = \text{Suc } (\text{Max } (?F \text{ } (cs @ [c]))) \text{ in}$   
 $y \in \text{sources-aux } (\text{drop } m \text{ } (cs @ [c])) (?G \text{ } m \text{ } (cs @ [c])) (?H \text{ } m \text{ } (cs @ [c])) f x$

*<proof>*

**lemma** *tags-sources*:

$(y, n) \in \text{tags } cs \text{ vs } s f x \implies$   
 $\text{let } m = \text{Suc } (\text{Max } \{k. k \leq \text{length } cs \wedge$   
 $\text{length } [c \leftarrow \text{take } k \text{ cs. } c = \text{IN } y] \leq n\})$   
 $\text{in } y \in \text{sources } (\text{drop } m \text{ cs}) \text{ (vs @ in-flow (take } m \text{ cs) vs } f)$   
 $(\text{run-flow (take } m \text{ cs) vs } s f) f x$

**and** *tags-aux-sources-aux*:

$(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies$   
 $\text{let } m = \text{Suc } (\text{Max } \{k. k \leq \text{length } cs \wedge$   
 $\text{length } [c \leftarrow \text{take } k \text{ cs. } c = \text{IN } y] \leq n\})$   
 $\text{in } y \in \text{sources-aux } (\text{drop } m \text{ cs}) \text{ (vs @ in-flow (take } m \text{ cs) vs } f)$   
 $(\text{run-flow (take } m \text{ cs) vs } s f) f x$

*<proof>*

**lemma** *tags-out-sources-out-1*:

**assumes**

$A: \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $(y, n) \in \text{tags-out } cs \text{ vs } s f x \implies$

$let\ m = Suc\ (Max\ \{k.\ k \leq\ length\ cs \wedge$   
 $\quad length\ [c \leftarrow take\ k\ cs.\ c = IN\ y] \leq\ n\})$   
 $in\ y \in sources-out\ (drop\ m\ cs)\ (vs\ @\ in-flow\ (take\ m\ cs)\ vs\ f)$   
 $\quad (run-flow\ (take\ m\ cs)\ vs\ s\ f)\ f\ x$   
**(is**  $\bigwedge - . - \implies - \implies let\ m = Suc\ (Max\ (?F\ cs))\ in$   
 $- \in sources-out\ -\ (?G\ m\ cs)\ (?H\ m\ cs)\ - -)$

**assumes**

**B:**  $\bigwedge z.\ c = (IN\ z :: com-flow) \implies$   
 $(y,\ n) \in tags-out\ cs\ vs\ s\ f\ x \implies let\ m = Suc\ (Max\ (?F\ cs))\ in$   
 $y \in sources-out\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$  **and**  
**C:**  $\bigwedge z.\ c = (OUT\ z :: com-flow) \implies$   
 $(y,\ n) \in tags-out\ cs\ vs\ s\ f\ x \implies let\ m = Suc\ (Max\ (?F\ cs))\ in$   
 $y \in sources-out\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$  **and**  
**D:**  $\bigwedge X.\ c = \langle X \rangle \implies$   
 $(y,\ n) \in tags-out\ cs\ vs\ s\ f\ x \implies let\ m = Suc\ (Max\ (?F\ cs))\ in$   
 $y \in sources-out\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$  **and**

**E:**  $(y,\ n) \in tags-out\ (cs\ @\ [c])\ vs\ s\ f\ x$

**shows**  $let\ m = Suc\ (Max\ (?F\ (cs\ @\ [c])))\ in$

$y \in sources-out\ (drop\ m\ (cs\ @\ [c]))\ (?G\ m\ (cs\ @\ [c]))\ (?H\ m\ (cs\ @\ [c]))\ f\ x$

$\langle proof \rangle$

**lemma** *tags-out-sources-out:*

$(y,\ n) \in tags-out\ cs\ vs\ s\ f\ x \implies$   
 $let\ m = Suc\ (Max\ \{k.\ k \leq\ length\ cs \wedge$   
 $\quad length\ [c \leftarrow take\ k\ cs.\ c = IN\ y] \leq\ n\})$   
 $in\ y \in sources-out\ (drop\ m\ cs)\ (vs\ @\ in-flow\ (take\ m\ cs)\ vs\ f)$   
 $\quad (run-flow\ (take\ m\ cs)\ vs\ s\ f)\ f\ x$

$\langle proof \rangle$

**lemma** *sources-member-1:*

**assumes**

**A:**  $\bigwedge z\ a.\ c = (z ::= a :: com-flow) \implies z = x \implies$   
 $y \in sources-aux\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ (cs\ @\ cs')\ vs\ s\ f\ x$   
**(is**  $\bigwedge - . - \implies - \implies - \in sources-aux\ -\ ?vs'\ ?s'\ - - \implies$   
 $- \subseteq sources-aux\ ?cs\ - - -)$

**assumes**

**B:**  $\bigwedge z\ a\ b\ w.\ c = (z ::= a :: com-flow) \implies z = x \implies$   
 $y \in sources\ cs'\ ?vs'\ ?s'\ f\ w \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ w$  **and**

**C:**  $\bigwedge z\ a.\ c = (z ::= a :: com-flow) \implies z \neq x \implies$   
 $y \in sources\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ x$  **and**

**D:**  $\bigwedge z.\ c = (IN\ z :: com-flow) \implies z = x \implies$   
 $y \in sources-aux\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ ?cs\ vs\ s\ f\ x$  **and**

**E:**  $\bigwedge z.\ c = (IN\ z :: com-flow) \implies z \neq x \implies$   
 $y \in sources\ cs'\ ?vs'\ ?s'\ f\ x \implies$

$sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ x$  **and**  
 $F: \bigwedge z. c = (OUT\ z :: com-flow) \implies$   
 $y \in sources\ cs' ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ x$  **and**  
 $G: \bigwedge X\ b\ w. c = \langle X \rangle \implies$   
 $y \in sources\ cs' ?vs' ?s' f\ w \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ w$   
**shows**  $y \in sources\ (cs' @ [c]) ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ (cs @ cs' @ [c])\ vs\ s\ f\ x$   
 $\langle proof \rangle$

**lemma** *sources-member-2:*

**assumes**

$A: \bigwedge z\ a. c = (z ::= a :: com-flow) \implies$   
 $y \in sources-aux\ cs' (vs @ in-flow\ cs\ vs\ f) (run-flow\ cs\ vs\ s\ f) f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ (cs @ cs')\ vs\ s\ f\ x$   
**(is**  $\bigwedge - . - \implies - \in sources-aux - ?vs' ?s' - - \implies$   
 $- \subseteq sources-aux\ ?cs - - - )$

**assumes**

$B: \bigwedge z. c = (IN\ z :: com-flow) \implies$   
 $y \in sources-aux\ cs' ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $C: \bigwedge z. c = (OUT\ z :: com-flow) \implies$   
 $y \in sources-aux\ cs' ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $D: \bigwedge X. c = \langle X \rangle \implies$   
 $y \in sources-aux\ cs' ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $E: \bigwedge X\ b\ w. c = \langle X \rangle \implies$   
 $y \in sources\ cs' ?vs' ?s' f\ w \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ ?cs\ vs\ s\ f\ w$   
**shows**  $y \in sources-aux\ (cs' @ [c]) ?vs' ?s' f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ (cs @ cs' @ [c])\ vs\ s\ f\ x$   
 $\langle proof \rangle$

**lemma** *sources-member:*

$y \in sources\ cs' (vs @ in-flow\ cs\ vs\ f) (run-flow\ cs\ vs\ s\ f) f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources\ (cs @ cs')\ vs\ s\ f\ x$   
**and** *sources-aux-member:*  
 $y \in sources-aux\ cs' (vs @ in-flow\ cs\ vs\ f) (run-flow\ cs\ vs\ s\ f) f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-aux\ (cs @ cs')\ vs\ s\ f\ x$   
 $\langle proof \rangle$

**lemma** *sources-out-member-1:*

**assumes**

$A: \bigwedge z\ a. c = (z ::= a :: com-flow) \implies$   
 $y \in sources-out\ cs' (vs @ in-flow\ cs\ vs\ f) (run-flow\ cs\ vs\ s\ f) f\ x \implies$   
 $sources\ cs\ vs\ s\ f\ y \subseteq sources-out\ (cs @ cs')\ vs\ s\ f\ x$

(is  $\bigwedge - . - \implies - \in \text{sources-out} - ?vs' ?s' - - \implies$   
 $- \subseteq \text{sources-out} ?cs - - - -$ )

**assumes**

*B*:  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies$   
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$   
 $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-out } ?cs \text{ vs } s f x$  **and**

*C*:  $\bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$   
 $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-out } ?cs \text{ vs } s f x$  **and**

*D*:  $\bigwedge X. c = \langle X \rangle \implies$   
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$   
 $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-out } ?cs \text{ vs } s f x$

**shows**  $y \in \text{sources-out } (cs' @ [c]) ?vs' ?s' f x \implies$   
 $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-out } (cs @ cs' @ [c]) \text{ vs } s f x$

*<proof>*

**lemma** *sources-out-member*:

$y \in \text{sources-out } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x \implies$   
 $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-out } (cs @ cs') \text{ vs } s f x$

*<proof>*

**lemma** *tags-member-1*:

**assumes**

*A*:  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z = x \implies$   
 $y \in \text{sources-aux } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-aux } (cs @ cs') \text{ vs } s f x$   
(is  $\bigwedge - . - \implies - \implies - \in \text{sources-aux} - ?vs' ?s' - - \implies$   
 $- \subseteq \text{tags-aux} ?cs - - - -$ )

**assumes**

*B*:  $\bigwedge z a b w. c = (z ::= a :: \text{com-flow}) \implies z = x \implies$   
 $y \in \text{sources } cs' ?vs' ?s' f w \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f w$  **and**

*C*:  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies z \neq x \implies$   
 $y \in \text{sources } cs' ?vs' ?s' f x \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f x$  **and**

*D*:  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies z = x \implies$   
 $y \in \text{sources-aux } cs' ?vs' ?s' f x \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-aux } ?cs \text{ vs } s f x$  **and**

*E*:  $\bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies z \neq x \implies$   
 $y \in \text{sources } cs' ?vs' ?s' f x \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f x$  **and**

*F*:  $\bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$   
 $y \in \text{sources } cs' ?vs' ?s' f x \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f x$  **and**

*G*:  $\bigwedge X b w. c = \langle X \rangle \implies$   
 $y \in \text{sources } cs' ?vs' ?s' f w \implies$   
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f w$

**shows**  $y \in \text{sources } (cs' @ [c]) ?vs' ?s' f x \implies$

$tags\ cs\ vs\ s\ f\ y \subseteq tags\ (cs\ @\ cs'\ @\ [c])\ vs\ s\ f\ x$   
 ⟨proof⟩

**lemma tags-member-2:**

**assumes**

$A: \bigwedge z\ a.\ c = (z ::= a :: com-flow) \implies$   
 $y \in sources-aux\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ (cs\ @\ cs')\ vs\ s\ f\ x$   
 (is  $\bigwedge - . - \implies - \in sources-aux - ?vs'\ ?s' - - \implies$   
 $- \subseteq tags-aux\ ?cs - - -$ )

**assumes**

$B: \bigwedge z.\ c = (IN\ z :: com-flow) \implies$   
 $y \in sources-aux\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $C: \bigwedge z.\ c = (OUT\ z :: com-flow) \implies$   
 $y \in sources-aux\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $D: \bigwedge X.\ c = \langle X \rangle \implies$   
 $y \in sources-aux\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ ?cs\ vs\ s\ f\ x$  **and**  
 $E: \bigwedge X\ b\ w.\ c = \langle X \rangle \implies$   
 $y \in sources\ cs'\ ?vs'\ ?s'\ f\ w \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags\ ?cs\ vs\ s\ f\ w$

**shows**  $y \in sources-aux\ (cs'\ @\ [c])\ ?vs'\ ?s'\ f\ x \implies$

$tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ (cs\ @\ cs'\ @\ [c])\ vs\ s\ f\ x$

⟨proof⟩

**lemma tags-member:**

$y \in sources\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags\ (cs\ @\ cs')\ vs\ s\ f\ x$

**and tags-aux-member:**

$y \in sources-aux\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-aux\ (cs\ @\ cs')\ vs\ s\ f\ x$

⟨proof⟩

**lemma tags-out-member-1:**

**assumes**

$A: \bigwedge z\ a.\ c = (z ::= a :: com-flow) \implies$   
 $y \in sources-out\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-out\ (cs\ @\ cs')\ vs\ s\ f\ x$   
 (is  $\bigwedge - . - \implies - \in sources-out - ?vs'\ ?s' - - \implies$   
 $- \subseteq tags-out\ ?cs - - -$ )

**assumes**

$B: \bigwedge z.\ c = (IN\ z :: com-flow) \implies$   
 $y \in sources-out\ cs'\ ?vs'\ ?s'\ f\ x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags-out\ ?cs\ vs\ s\ f\ x$  **and**  
 $C: \bigwedge z.\ c = (OUT\ z :: com-flow) \implies$   
 $y \in sources-out\ cs'\ ?vs'\ ?s'\ f\ x \implies$

$tags\ cs\ vs\ s\ f\ y \subseteq tags\text{-out}\ ?cs\ vs\ s\ f\ x$  **and**  
 $D: \bigwedge X. c = \langle X \rangle \implies$   
 $y \in sources\text{-out}\ cs' ?vs' ?s' f x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags\text{-out}\ ?cs\ vs\ s\ f\ x$   
**shows**  $y \in sources\text{-out}\ (cs' @ [c]) ?vs' ?s' f x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags\text{-out}\ (cs @ cs' @ [c]) vs\ s\ f\ x$   
 $\langle proof \rangle$

**lemma** *tags-out-member*:

$y \in sources\text{-out}\ cs' (vs @ in\text{-flow}\ cs\ vs\ f) (run\text{-flow}\ cs\ vs\ s\ f) f x \implies$   
 $tags\ cs\ vs\ s\ f\ y \subseteq tags\text{-out}\ (cs @ cs') vs\ s\ f\ x$   
 $\langle proof \rangle$

**lemma** *tags-suffix-1*:

**assumes**

$A: \bigwedge z\ a. c = (z ::= a :: com\text{-flow}) \implies z = x \implies$   
 $tags\text{-aux}\ cs' (vs @ in\text{-flow}\ cs\ vs\ f) (run\text{-flow}\ cs\ vs\ s\ f) f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\text{-aux}\ (cs @ cs') vs\ s\ f\ x\}$   
 $(is\ \bigwedge - . - \implies - \implies tags\text{-aux}\ - ?vs' ?s' - - = -)$

**assumes**

$B: \bigwedge z\ a\ b\ y. c = (z ::= a :: com\text{-flow}) \implies z = x \implies$   
 $tags\ cs' ?vs' ?s' f y =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs') vs\ s\ f\ y\}$  **and**

$C: \bigwedge z\ a. c = (z ::= a :: com\text{-flow}) \implies z \neq x \implies$   
 $tags\ cs' ?vs' ?s' f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs') vs\ s\ f\ x\}$  **and**

$D: \bigwedge z. c = (IN\ z :: com\text{-flow}) \implies z = x \implies$   
 $tags\text{-aux}\ cs' ?vs' ?s' f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\text{-aux}\ (cs @ cs') vs\ s\ f\ x\}$  **and**

$E: \bigwedge z. c = (IN\ z :: com\text{-flow}) \implies z \neq x \implies$   
 $tags\ cs' ?vs' ?s' f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs') vs\ s\ f\ x\}$  **and**

$F: \bigwedge z. c = (OUT\ z :: com\text{-flow}) \implies$   
 $tags\ cs' ?vs' ?s' f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs') vs\ s\ f\ x\}$  **and**

$G: \bigwedge X\ b\ y. c = \langle X \rangle \implies$   
 $tags\ cs' ?vs' ?s' f y =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs') vs\ s\ f\ y\}$

**shows**  $tags\ (cs' @ [c]) ?vs' ?s' f x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in tags\ (cs @ cs' @ [c]) vs\ s\ f\ x\}$

(is - = {p. case p of (w, n) ⇒ ?P w n c})  
 ⟨proof⟩

**lemma tags-suffix-2:**

**assumes**

A:  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $\text{tags-aux } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$   
 (is  $\bigwedge - . - \implies \text{tags-aux } - ?vs' ?s' - - = -$ )

**assumes**

B:  $\bigwedge z. c = (IN z :: \text{com-flow}) \implies$   
 $\text{tags-aux } cs' ?vs' ?s' f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$  **and**

C:  $\bigwedge z. c = (OUT z :: \text{com-flow}) \implies$   
 $\text{tags-aux } cs' ?vs' ?s' f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$  **and**

D:  $\bigwedge X. c = \langle X \rangle \implies$   
 $\text{tags-aux } cs' ?vs' ?s' f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$  **and**

E:  $\bigwedge X b y. c = \langle X \rangle \implies$   
 $\text{tags } cs' ?vs' ?s' f y =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags } (cs @ cs') \text{ vs } s f y\}$

**shows**  $\text{tags-aux } (cs' @ [c]) ?vs' ?s' f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-aux } (cs @ cs' @ [c]) \text{ vs } s f x\}$   
 (is - = {p. case p of (w, n) ⇒ ?P w n c})  
 ⟨proof⟩

**lemma tags-suffix:**

$\text{tags } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x = \{(w, n).$   
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags } (cs @ cs') \text{ vs } s f x\}$

**and tags-aux-suffix:**

$\text{tags-aux } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x = \{(w, n).$   
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$   
 ⟨proof⟩

**lemma tags-out-suffix-1:**

**assumes**

A:  $\bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$   
 $\text{tags-out } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x =$   
 $\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$   
 $\in \text{tags-out } (cs @ cs') \text{ vs } s f x\}$   
 (is  $\bigwedge - . - \implies \text{tags-out } - ?vs' ?s' - - = -$ )

**assumes**

$B: \bigwedge z. c = (IN\ z :: com-flow) \implies$   
 $tags-out\ cs'\ ?vs'\ ?s'\ f\ x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in\ tags-out\ (cs\ @\ cs')\ vs\ s\ f\ x\} \text{ and}$

$C: \bigwedge z. c = (OUT\ z :: com-flow) \implies$   
 $tags-out\ cs'\ ?vs'\ ?s'\ f\ x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in\ tags-out\ (cs\ @\ cs')\ vs\ s\ f\ x\} \text{ and}$

$D: \bigwedge X. c = \langle X \rangle \implies$   
 $tags-out\ cs'\ ?vs'\ ?s'\ f\ x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in\ tags-out\ (cs\ @\ cs')\ vs\ s\ f\ x\}$

**shows**  $tags-out\ (cs'\ @\ [c])\ ?vs'\ ?s'\ f\ x =$   
 $\{p. case\ p\ of\ (w, n) \Rightarrow (w, length\ [c \leftarrow cs. c = IN\ w] + n)$   
 $\in\ tags-out\ (cs\ @\ cs'\ @\ [c])\ vs\ s\ f\ x\}$   
**(is - =**  $\{p. case\ p\ of\ (w, n) \Rightarrow ?P\ w\ n\ c\}$ )  
 $\langle proof \rangle$

**lemma** *tags-out-suffix*:

$tags-out\ cs'\ (vs\ @\ in-flow\ cs\ vs\ f)\ (run-flow\ cs\ vs\ s\ f)\ f\ x = \{(w, n).$   
 $(w, length\ [c \leftarrow cs. c = IN\ w] + n) \in\ tags-out\ (cs\ @\ cs')\ vs\ s\ f\ x\}$   
 $\langle proof \rangle$

**lemma** *sources-aux-rhs*:

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq sources-aux\ (flow\ cfs\ @\ cs')\ vs_1\ s_1\ f\ x)\}$   
**(is -  $\subseteq$  {-. - = - ( $\subseteq sources-aux\ (?cs\ @\ -) - - -$ )})**

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{tags-aux\ (?cs\ @\ cs')\ vs_1\ s_1\ f\ x \mid x. x \in S\}) \text{ and}$   
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow * \{cfs\} (c_2, s_2, f, vs_2, ws_2) \text{ and}$   
 $D: ok-flow-aux-2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs$

**shows**  $S \subseteq \{x. s_2 = t_2 (\subseteq sources-aux\ cs'\ vs_2\ s_2\ f\ x)\}$   
 $\langle proof \rangle$

**lemma** *sources-rhs*:

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq sources\ (flow\ cfs\ @\ cs')\ vs_1\ s_1\ f\ x)\}$   
**(is -  $\subseteq$  {-. - = - ( $\subseteq sources\ (?cs\ @\ -) - - -$ )})**

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{tags\ (?cs\ @\ cs')\ vs_1\ s_1\ f\ x \mid x. x \in S\}) \text{ and}$   
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow * \{cfs\} (c_2, s_2, f, vs_2, ws_2) \text{ and}$   
 $D: ok-flow-aux-2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs$

**shows**  $S \subseteq \{x. s_2 = t_2 (\subseteq sources\ cs'\ vs_2\ s_2\ f\ x)\}$   
 $\langle proof \rangle$

**lemma sources-out-rhs:**

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out (flow cfs @ cs')} vs_1 s_1 f x)\}$   
 $(\text{is } \subseteq \{-. - = - (\subseteq \text{sources-out (?cs @ -) - - -})\})$

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{\text{tags-out (?cs @ cs')} vs_1 s_1 f x \mid x. x \in S\})$  **and**  
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
 $D: \text{ok-flow-aux-2 } s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs$

**shows**  $S \subseteq \{x. s_2 = t_2 (\subseteq \text{sources-out cs' vs}_2 s_2 f x)\}$   
 $\langle \text{proof} \rangle$

**lemma tags-aux-rhs:**

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux (flow cfs @ cs')} vs_1 s_1 f x)\}$   
 $(\text{is } \subseteq \{-. - = - (\subseteq \text{sources-aux (?cs @ -) - - -})\})$

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{\text{tags-aux (?cs @ cs')} vs_1 s_1 f x \mid x. x \in S\})$  **and**  
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
 $D: (c_1', t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$  **and**  
 $E: \text{ok-flow-aux-1 } c_1 c_2 c_2' s_1 t_1 t_2 f f' vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs$

**shows**  $f = f' (\subseteq vs_2, vs_2', \bigcup \{\text{tags-aux cs' vs}_2 s_2 f x \mid x. x \in S\})$   
 $\langle \text{proof} \rangle$

**lemma tags-rhs:**

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources (flow cfs @ cs')} vs_1 s_1 f x)\}$   
 $(\text{is } \subseteq \{-. - = - (\subseteq \text{sources (?cs @ -) - - -})\})$

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{\text{tags (?cs @ cs')} vs_1 s_1 f x \mid x. x \in S\})$  **and**  
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
 $D: (c_1', t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$  **and**  
 $E: \text{ok-flow-aux-1 } c_1 c_2 c_2' s_1 t_1 t_2 f f' vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs$

**shows**  $f = f' (\subseteq vs_2, vs_2', \bigcup \{\text{tags cs' vs}_2 s_2 f x \mid x. x \in S\})$   
 $\langle \text{proof} \rangle$

**lemma tags-out-rhs:**

**assumes**

$A: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out (flow cfs @ cs')} vs_1 s_1 f x)\}$   
 $(\text{is } \subseteq \{-. - = - (\subseteq \text{sources-out (?cs @ -) - - -})\})$

**assumes**

$B: f = f' (\subseteq vs_1, vs_1',$   
 $\bigcup \{\text{tags-out (?cs @ cs')} vs_1 s_1 f x \mid x. x \in S\})$  **and**  
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$  **and**  
 $D: (c_1', t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$  **and**  
 $E: \text{ok-flow-aux-1 } c_1 c_2 c_2' s_1 t_1 t_2 f f' vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs$

**shows**  $f = f' (\subseteq vs_2, vs_2', \cup \{tags-out\ cs' vs_2 s_2 f x \mid x. x \in S\})$   
 ⟨proof⟩

**lemma** *ctyping2-term-seq*:

**assumes**

$A: \bigwedge B Y p. (U, v) \models c_1 (\subseteq A, X) = Some (B, Y) \implies$   
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV \implies \exists p'. (c_1, p) \Rightarrow p'$  **and**  
 $B: \bigwedge q B Y B' Y' p. (U, v) \models c_1 (\subseteq A, X) = Some q \implies (B, Y) = q \implies$   
 $(U, v) \models c_2 (\subseteq B, Y) = Some (B', Y') \implies$   
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV \implies \exists p'. (c_2, p) \Rightarrow p'$  **and**  
 $C: (U, v) \models c_1;; c_2 (\subseteq A, X) = Some (B', Y')$  **and**  
 $D: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV$

**shows**  $\exists p'. (c_1;; c_2, p) \Rightarrow p'$

⟨proof⟩

**lemma** *ctyping2-term-or*:

**assumes**

$A: \bigwedge B Y p. (U, v) \models c_1 (\subseteq A, X) = Some (B, Y) \implies$   
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV \implies \exists p'. (c_1, p) \Rightarrow p'$  **and**  
 $B: (U, v) \models c_1 OR c_2 (\subseteq A, X) = Some (B', Y')$  **and**  
 $C: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV$

**shows**  $\exists p'. (c_1 OR c_2, p) \Rightarrow p'$

⟨proof⟩

**lemma** *ctyping2-term-if*:

**assumes**

$A: \bigwedge U' q B_1 B_2 B Y p.$   
 $(U', q) = (insert (Univ? A X, bvars b) U, \models b (\subseteq A, X)) \implies$   
 $(B_1, B_2) = q \implies (U', v) \models c_1 (\subseteq B_1, X) = Some (B, Y) \implies$   
 $\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow UNIV \implies \exists p'. (c_1, p) \Rightarrow p'$  **and**  
 $B: \bigwedge U' q B_1 B_2 B Y p.$   
 $(U', q) = (insert (Univ? A X, bvars b) U, \models b (\subseteq A, X)) \implies$   
 $(B_1, B_2) = q \implies (U', v) \models c_2 (\subseteq B_2, X) = Some (B, Y) \implies$   
 $\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow UNIV \implies \exists p'. (c_2, p) \Rightarrow p'$  **and**  
 $C: (U, v) \models IF b THEN c_1 ELSE c_2 (\subseteq A, X) = Some (B, Y)$  **and**  
 $D: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV$

**shows**  $\exists p'. (IF b THEN c_1 ELSE c_2, p) \Rightarrow p'$

⟨proof⟩

**lemma** *ctyping2-term*:

$\llbracket (U, v) \models c (\subseteq A, X) = Some (B, Y); \exists (C, Z) \in U. \neg C: Z \rightsquigarrow UNIV \rrbracket \implies$   
 $\exists p'. (c, p) \Rightarrow p'$

⟨proof⟩

**lemma** *ctyping2-confine-seq*:

**assumes**

$A: \bigwedge s' f' vs' ws' A B X Y U v. p = (s', f', vs', ws') \implies$

$(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies \exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$   
 $s = s' (\subseteq S) \wedge$   
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ } vs'. \text{fst } p \in S] = [] \wedge$   
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ } ws'. \text{fst } p \in S] = []$   
**(is**  $\bigwedge s' - vs' ws' \text{ } \dots \implies \dots \implies \dots \implies$   
 $?P \text{ } s \text{ } s' \text{ } vs \text{ } vs' \text{ } ws \text{ } ws')$

**assumes**

$B: \bigwedge s' f' vs' ws' A B X Y U v. p = (s', f', vs', ws') \implies$   
 $(U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies \exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$   
 $?P \text{ } s' \text{ } s'' \text{ } vs' \text{ } vs'' \text{ } ws' \text{ } ws''$  **and**  
 $C: (c_1, s, f, vs, ws) \Rightarrow p$  **and**  
 $D: (c_2, p) \Rightarrow (s'', f'', vs'', ws'')$  **and**  
 $E: (U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (B', Y')$  **and**  
 $F: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$   
**shows**  $?P \text{ } s \text{ } s'' \text{ } vs \text{ } vs'' \text{ } ws \text{ } ws''$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-confine-or-lhs*:

**assumes**

$A: \bigwedge A B X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$   
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$   
 $s = s' (\subseteq S) \wedge$   
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ } vs'. \text{fst } p \in S] = [] \wedge$   
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ } ws'. \text{fst } p \in S] = []$   
**(is**  $\bigwedge \dots \implies \dots \implies ?P)$

**assumes**

$B: (U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B', Y')$  **and**  
 $C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

**shows**  $?P$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-confine-or-rhs*:

**assumes**

$A: \bigwedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies$   
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$   
 $s = s' (\subseteq S) \wedge$   
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ } vs'. \text{fst } p \in S] = [] \wedge$   
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ } ws'. \text{fst } p \in S] = []$   
**(is**  $\bigwedge \dots \implies \dots \implies ?P)$

**assumes**

$B: (U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B', Y')$  **and**  
 $C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

**shows**  $?P$

$\langle \text{proof} \rangle$

**lemma** *ctyping2-confine-if-true*:

**assumes**

$A: \bigwedge A B X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$   
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$

$$s = s' (\subseteq S) \wedge$$

$$[p \leftarrow \text{drop} (\text{length } vs) \text{ vs}'. \text{fst } p \in S] = [] \wedge$$

$$[p \leftarrow \text{drop} (\text{length } ws) \text{ ws}'. \text{fst } p \in S] = []$$

(is  $\bigwedge \text{---} \implies \text{---} \implies ?P$ )

**assumes**

$B: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y)$  **and**

$C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

**shows**  $?P$

*<proof>*

**lemma** *ctyping2-confine-if-false:*

**assumes**

$A: \bigwedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies$

$\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$

$s = s' (\subseteq S) \wedge$

$[p \leftarrow \text{drop} (\text{length } vs) \text{ vs}'. \text{fst } p \in S] = [] \wedge$

$[p \leftarrow \text{drop} (\text{length } ws) \text{ ws}'. \text{fst } p \in S] = []$

(is  $\bigwedge \text{---} \implies \text{---} \implies ?P$ )

**assumes**

$B: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y)$  **and**

$C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

**shows**  $?P$

*<proof>*

**lemma** *ctyping2-confine:*

$\llbracket (c, s, f, vs, ws) \Rightarrow (s', f', vs', ws') \rrbracket;$

$(U, v) \models c (\subseteq A, X) = \text{Some } (B, Y); \exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$

$s = s' (\subseteq S) \wedge$

$[p \leftarrow \text{drop} (\text{length } vs) \text{ vs}'. \text{fst } p \in S] = [] \wedge$

$[p \leftarrow \text{drop} (\text{length } ws) \text{ ws}'. \text{fst } p \in S] = []$

(is  $\llbracket \text{---}; \text{---}; \text{---} \rrbracket \implies ?P \text{ s } s' \text{ vs } vs' \text{ ws } ws'$ )

*<proof>*

**lemma** *eq-states-assign:*

**assumes**

$A: S \subseteq \{y. s = t (\subseteq \text{sources } [x ::= a] \text{ vs } s f y)\}$  **and**

$B: x \in S$  **and**

$C: s \in \text{Univ } A (\subseteq \text{state} \cap X)$  **and**

$D: \text{Univ? } A X: \text{avars } a \rightsquigarrow \{x\}$

**shows**  $s = t (\subseteq \text{avars } a)$

*<proof>*

**lemma** *eq-states-while:*

**assumes**

$A: S \subseteq \{x. s = t (\subseteq \text{sources-aux } ((\text{bvars } b) \# \text{cs}) \text{ vs } s f x)\}$  **and**

$B: S \neq \{\}$  **and**

$C: s \in \text{Univ } A (\subseteq \text{state} \cap X) \cup \text{Univ } C (\subseteq \text{state} \cap Y)$  **and**

$D: \text{Univ? } A X \cup \text{Univ? } C Y: \text{bvars } b \rightsquigarrow \text{UNIV}$

**shows**  $s = t (\subseteq \text{bvars } b)$   
 $\langle \text{proof} \rangle$

**lemma** *univ-states-while*:

**assumes**

$A: (c, s, p) \Rightarrow (s', p')$  **and**

$B: \models b (\subseteq A, X) = (B_1, B_2)$  **and**

$C: \vdash c (\subseteq B_1, X) = (C, Y)$  **and**

$D: \models b (\subseteq C, Y) = (B_1', B_2')$  **and**

$E: (\{\}, \text{False}) \models c (\subseteq B_1, X) = \text{Some } (D, Z)$  **and**

$F: (\{\}, \text{False}) \models c (\subseteq B_1', Y) = \text{Some } (D', Z')$  **and**

$G: \text{bval } b \ s$

**shows**  $s \in \text{Univ } A (\subseteq \text{state} \cap X) \cup \text{Univ } C (\subseteq \text{state} \cap Y) \Longrightarrow$

$s' \in \text{Univ } A (\subseteq \text{state} \cap X) \cup \text{Univ } C (\subseteq \text{state} \cap Y)$

$\langle \text{proof} \rangle$

**end**

**end**

## 6 Sufficiency of well-typedness for information flow correctness: main theorem

**theory** *Correctness-Theorem*

**imports** *Correctness-Lemmas*

**begin**

The purpose of this section is to prove that type system *ctyping2* is correct in that it guarantees that well-typed programs satisfy the information flow correctness criterion expressed by predicate *correct*, namely that if the type system outputs a value other than *None* (that is, a *pass* verdict) when it is input program  $c$ , *state set*  $A$ , and *vname set*  $X$ , then  $\text{correct } c \ A \ X$  (theorem *ctyping2-correct*).

This proof makes use of the lemma *ctyping2-approx* proven in a previous section.

### 6.1 Local context proofs

**context** *noninterf*

**begin**

**lemma** *ctyping2-correct-aux-skip* [*elim!*]:

$\llbracket (\text{SKIP}, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c_1, s_1, f, vs_1, ws_1);$   
 $(c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2) \rrbracket \Longrightarrow$   
 $ok\text{-flow-aux } U \ c_1 \ c_2 \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ (\text{flow } cfs_2)$

*<proof>*

**lemma** *ctyping2-correct-aux-assign*:

**assumes**

$A: (U, v) \models x ::= a (\subseteq A, X) = \text{Some } (C, Y)$  **and**

$B: s \in \text{Univ } A (\subseteq \text{state} \cap X)$  **and**

$C: (x ::= a, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$  **and**

$D: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*)

*<proof>*

**lemma** *ctyping2-correct-aux-input*:

**assumes**

$A: (U, v) \models \text{IN } x (\subseteq A, X) = \text{Some } (C, Y)$  **and**

$B: (\text{IN } x, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$  **and**

$C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*)

*<proof>*

**lemma** *ctyping2-correct-aux-output*:

**assumes**

$A: (U, v) \models \text{OUT } x (\subseteq A, X) = \text{Some } (B, Y)$  **and**

$B: (\text{OUT } x, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$  **and**

$C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*)

*<proof>*

**lemma** *ctyping2-correct-aux-seq*:

**assumes**

$A: (U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (C, Z)$  **and**

$B: \bigwedge B Y c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$

$s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies$

$(c_1, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*) **and**

$C: \bigwedge p B Y C Z c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U, v) \models c_1 (\subseteq A, X) = \text{Some } p \implies (B, Y) = p \implies$

$(U, v) \models c_2 (\subseteq B, Y) = \text{Some } (C, Z) \implies$

$s \in \text{Univ } B (\subseteq \text{state} \cap Y) \implies$

$(c_2, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*) **and**

$D: s \in \text{Univ } A (\subseteq \text{state} \cap X)$  **and**

$E: (c_1;; c_2, s, f, vs_0, ws_0) \rightarrow^* \{cfs_1\} (c', s_1, f, vs_1, ws_1)$  **and**

$F: (c', s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c'', s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$  (*flow cfs<sub>2</sub>*)

*<proof>*

**lemma** *ctyping2-correct-aux-or*:

**assumes**

$A: (U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C, Y) \text{ and}$

$B: \bigwedge C Y c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U, v) \models c_1 (\subseteq A, X) = \text{Some } (C, Y) \implies$

$s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies$

$(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2) \text{ and}$

$C: \bigwedge C Y c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U, v) \models c_2 (\subseteq A, X) = \text{Some } (C, Y) \implies$

$s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies$

$(c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2) \text{ and}$

$D: s \in \text{Univ } A (\subseteq \text{state} \cap X) \text{ and}$

$E: (c_1 \text{ OR } c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \text{ and}$

$F: (c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2)$

*<proof>*

**lemma** *ctyping2-correct-aux-if*:

**assumes**

$A: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (C, Y) \text{ and}$

$B: \bigwedge U' p B_1 B_2 C_1 Y_1 c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$

$(B_1, B_2) = p \implies$

$(U', v) \models c_1 (\subseteq B_1, X) = \text{Some } (C_1, Y_1) \implies$

$s \in \text{Univ } B_1 (\subseteq \text{state} \cap X) \implies$

$(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U' c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2) \text{ and}$

$C: \bigwedge U' p B_1 B_2 C_2 Y_2 c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

$(U', p) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$

$(B_1, B_2) = p \implies$

$(U', v) \models c_2 (\subseteq B_2, X) = \text{Some } (C_2, Y_2) \implies$

$s \in \text{Univ } B_2 (\subseteq \text{state} \cap X) \implies$

$(c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$

$(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$

*ok-flow-aux*  $U' c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2) \text{ and}$

$D: s \in \text{Univ } A (\subseteq \text{state} \cap X) \text{ and}$

$E: (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\}$

$(c', s_1, f, vs_1, ws_1) \text{ and}$

$F: (c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2)$

**shows** *ok-flow-aux*  $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2 (\text{flow } cfs_2)$

*<proof>*

**lemma** *ctyping2-correct-aux-while*:

**assumes**

**A:**  $(U, v) \models \text{WHILE } b \text{ DO } c \ (\subseteq A, X) = \text{Some } (B, W) \text{ and}$   
**B:**  $\bigwedge B_1 B_2 C Y B_1' B_2' D Z c_1 c_2 s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$   
 $(B_1, B_2) = \models b \ (\subseteq A, X) \implies$   
 $(C, Y) = \vdash c \ (\subseteq B_1, X) \implies$   
 $(B_1', B_2') = \models b \ (\subseteq C, Y) \implies$   
 $\forall (B, W) \in \text{insert } (\text{Univ? } A X \cup \text{Univ? } C Y, \text{bvars } b) U.$   
**B:**  $W \rightsquigarrow \text{UNIV} \implies$   
 $(\{\}, \text{False}) \models c \ (\subseteq B_1, X) = \text{Some } (D, Z) \implies$   
 $s \in \text{Univ } B_1 \ (\subseteq \text{state} \cap X) \implies$   
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1) \implies$   
 $(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2) \implies$   
 $\text{ok-flow-aux } \{\} c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2 \ (\text{flow } cfs_2) \text{ and}$   
**C:**  $\bigwedge B_1 B_2 C Y B_1' B_2' D' Z' c_1 c_2 s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$   
 $(B_1, B_2) = \models b \ (\subseteq A, X) \implies$   
 $(C, Y) = \vdash c \ (\subseteq B_1, X) \implies$   
 $(B_1', B_2') = \models b \ (\subseteq C, Y) \implies$   
 $\forall (B, W) \in \text{insert } (\text{Univ? } A X \cup \text{Univ? } C Y, \text{bvars } b) U.$   
**B:**  $W \rightsquigarrow \text{UNIV} \implies$   
 $(\{\}, \text{False}) \models c \ (\subseteq B_1', Y) = \text{Some } (D', Z') \implies$   
 $s \in \text{Univ } B_1' \ (\subseteq \text{state} \cap Y) \implies$   
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1) \implies$   
 $(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2) \implies$   
 $\text{ok-flow-aux } \{\} c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2 \ (\text{flow } cfs_2) \text{ and}$   
**D:**  $s \in \text{Univ } A \ (\subseteq \text{state} \cap X) \text{ and}$   
**E:**  $(\text{WHILE } b \text{ DO } c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1) \text{ and}$   
**F:**  $(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$   
**shows**  $\text{ok-flow-aux } U c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2 \ (\text{flow } cfs_2)$   
 $\langle \text{proof} \rangle$

**lemma** *ctyping2-correct-aux:*

$\llbracket (U, v) \models c \ (\subseteq A, X) = \text{Some } (B, Y); s \in \text{Univ } A \ (\subseteq \text{state} \cap X);$   
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1);$   
 $(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2) \rrbracket \implies$   
 $\text{ok-flow-aux } U c_1 c_2 s_1 s_2 f vs_1 vs_2 ws_1 ws_2 \ (\text{flow } cfs_2)$   
 $\langle \text{proof} \rangle$

**theorem** *ctyping2-correct:*

**assumes**  $A: (U, v) \models c \ (\subseteq A, X) = \text{Some } (B, Y)$   
**shows**  $\text{correct } c A X$   
 $\langle \text{proof} \rangle$

**end**

**end**

## References

- [1] C. Ballarin. *Tutorial to Locales and Locale Interpretation*. <https://isabelle.in.tum.de/website-Isabelle2024/dist/Isabelle2024/doc/locales.pdf>.
- [2] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. <https://isabelle.in.tum.de/website-Isabelle2024/dist/Isabelle2024/doc/functions.pdf>.
- [3] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs*. <https://isabelle.in.tum.de/website-Isabelle2011/dist/Isabelle2011/doc/isar-overview.pdf>.
- [4] T. Nipkow. *Programming and Proving in Isabelle/HOL*, May 2024. <https://isabelle.in.tum.de/website-Isabelle2024/dist/Isabelle2024/doc/prog-prove.pdf>.
- [5] T. Nipkow and G. Klein. Theory HOL-IMP.Big\_Step (included in the Isabelle2024 distribution). [https://isabelle.in.tum.de/website-Isabelle2024/dist/library/HOL/HOL-IMP/Big\\_Step.html](https://isabelle.in.tum.de/website-Isabelle2024/dist/library/HOL/HOL-IMP/Big_Step.html).
- [6] T. Nipkow and G. Klein. Theory HOL-IMP.Com (included in the Isabelle2024 distribution). <https://isabelle.in.tum.de/website-Isabelle2024/dist/library/HOL/HOL-IMP/Com.html>.
- [7] T. Nipkow and G. Klein. Theory HOL-IMP.Small\_Step (included in the Isabelle2024 distribution). [https://isabelle.in.tum.de/website-Isabelle2024/dist/library/HOL/HOL-IMP/Small\\_Step.html](https://isabelle.in.tum.de/website-Isabelle2024/dist/library/HOL/HOL-IMP/Small_Step.html).
- [8] T. Nipkow and G. Klein. *Concrete Semantics with Isabelle/HOL*. Springer-Verlag, Feb. 2023. (Current version: <http://www.concrete-semantics.org/concrete-semantics.pdf>).
- [9] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, May 2024. <https://isabelle.in.tum.de/website-Isabelle2024/dist/Isabelle2024/doc/tutorial.pdf>.
- [10] P. Noce. Information Flow Control via Stateful Intransitive Noninterference in Language IMP. *Archive of Formal Proofs*, Feb. 2024. [https://isa-afp.org/entries/IMP\\_Noninterference.html](https://isa-afp.org/entries/IMP_Noninterference.html), Formal proof development.
- [11] D. Volpano and G. Smith. Eliminating Covert Flows with Minimum Typings. In *Proc. 10th IEEE Computer Security Foundations Workshop*, June 1997.

- [12] D. Volpano, G. Smith, and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, Jan. 1996.