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

September 3, 2024

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

datatype *com* =
SKIP |
Assign vname aexp (- ::= - [*1000*, *61*] *70*) |
Input vname ((*IN* -) [*61*] *70*) |
Output vname ((*OUT* -) [*61*] *70*) |
Seq com com (-;;/ - [*61*, *61*] *70*) |
Or com com ((- *OR* -) [*61*, *61*] *70*) |
If bexp com com ((*IF* -/ *THEN* -/ *ELSE* -) [*0*, *0*, *61*] *70*) |
While bexp com ((*WHILE* -/ *DO* -) [*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* ⇒ *nat* ⇒ *val*
type-synonym *inputs* = (*vname* × *val*) *list*
type-synonym *outputs* = (*vname* × *val*) *list*

type-synonym *stage* = *state* × *stream* × *inputs* × *outputs*

inductive *big-step* :: *com* × *stage* ⇒ *stage* ⇒ *bool*
(**infix** ⇒ 55) **where**
Skip:
(*SKIP*, *p*) ⇒ *p* |
Assign:
(*x* ::= *a*, *s*, *p*) ⇒ (*s*(*x* := *aval a s*), *p*) |
Input:
n = *length* [*p*←*vs*. *fst p* = *x*] ⟹ (*IN x*, *s*, *f*, *vs*, *ws*) ⇒
(*s*(*x* := *f x n*), *f*, *vs* @ [(*x*, *f x n*)], *ws*) |
Output:
(*OUT x*, *s*, *f*, *vs*, *ws*) ⇒ (*s*, *f*, *vs*, *ws* @ [(*x*, *s x*)]) |
Seq:
⟦(*c*1, *p*1) ⇒ *p*2; (*c*2, *p*2) ⇒ *p*3⟧ ⟹ (*c*1;; *c*2, *p*1) ⇒ *p*3 |
Or1:
(*c*1, *p*) ⇒ *p*′ ⟹ (*c*1 *OR c*2, *p*) ⇒ *p*′ |
Or2:
(*c*2, *p*) ⇒ *p*′ ⟹ (*c*1 *OR c*2, *p*) ⇒ *p*′ |
IfTrue:
⟦*bval b s*; (*c*1, *s*, *p*) ⇒ *p*′⟧ ⟹
(*IF b THEN c*1 *ELSE c*2, *s*, *p*) ⇒ *p*′ |
IfFalse:
⟦¬ *bval b s*; (*c*2, *s*, *p*) ⇒ *p*′⟧ ⟹
(*IF b THEN c*1 *ELSE c*2, *s*, *p*) ⇒ *p*′ |
WhileFalse:
¬ *bval b s* ⟹ (*WHILE b DO c*, *s*, *p*) ⇒ (*s*, *p*) |
WhileTrue:
⟦*bval b s*1; (*c*, *s*1, *p*1) ⇒ (*s*2, *p*2);
(*WHILE b DO c*, *s*2, *p*2) ⇒ (*s*3, *p*3)⟧ ⟹
(*WHILE b DO c*, *s*1, *p*1) ⇒ (*s*3, *p*3)

declare *big-step.intros* [*intro*]

inductive-cases *SkipE* [*elim*!]: (*SKIP*, *p*) ⇒ *p*′

inductive-cases *AssignE* [*elim*!]: (*x* ::= *a*, *p*) ⇒ *p*′

inductive-cases *InputE* [*elim*!]: (*IN x*, *p*) ⇒ *p*′

inductive-cases *OutputE* [*elim*!]: (*OUT x*, *p*) ⇒ *p*′

inductive-cases *SeqE* [*elim*!]: (*c*1;; *c*2, *p*) ⇒ *p*′

inductive-cases *OrE* [*elim*!]: (*c*1 *OR c*2, *p*) ⇒ *p*′

inductive-cases *IfE* [*elim*!]: (*IF b THEN c*1 *ELSE c*2, *p*) ⇒ *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** $\rightarrow$ *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:
$(SKIP;;\ c_2,\ p) \rightarrow (c_2,\ p)$ |
Seq2:
$(c_1,\ p) \rightarrow (c_1{}',\ p') \Longrightarrow (c_1;;\ c_2,\ p) \rightarrow (c_1{}';;\ c_2,\ p')$ |
Or1:
$(c_1\ OR\ c_2,\ p) \rightarrow (c_1,\ p)$ |
Or2:
$(c_1\ OR\ c_2,\ p) \rightarrow (c_2,\ p)$ |
IfTrue:
$bval\ b\ s \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s,\ p) \rightarrow (c_1,\ s,\ p)$ |
IfFalse:
$\neg\ bval\ b\ s \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s,\ p) \rightarrow (c_2,\ s,\ p)$ |
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*!]: $(c_1;;\ c_2,\ p) \rightarrow cf$

inductive-cases *orE* [*elim*!]: $(c_1\ OR\ c_2,\ p) \rightarrow cf$

inductive-cases *ifE* [*elim*!]: $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ p) \rightarrow cf$

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

abbreviation *small-steps* :: $com \times stage \Rightarrow com \times stage \Rightarrow bool$
(**infix** $\rightarrow*$ 55) **where**
$cf \rightarrow* cf' \equiv star\ small\text{-}step\ cf\ cf'$

function *small-stepsl* ::
$com \times stage \Rightarrow (com \times stage)\ list \Rightarrow com \times stage \Rightarrow bool$
$((\text{-} \rightarrow*'\{\text{-}'\}\ \text{-})\ [51,\ 51]\ 55)$
where
$cf \rightarrow*\{[]\}\ cf' = (cf = cf')\ |$
$cf \rightarrow*\{cfs\ @\ [cf']\}\ cf'' = (cf \rightarrow*\{cfs\}\ cf' \wedge cf' \rightarrow cf'')$

⟨*proof*⟩
termination ⟨*proof*⟩

1.4 Equivalence of big-step and small-step semantics

lemma *star-seq2*:
$(c_1,\ p) \rightarrow* (c_1',\ p') \Longrightarrow (c_1;;\ c_2,\ p) \rightarrow* (c_1';;\ c_2,\ p')$
⟨*proof*⟩

lemma *seq-comp*:
$[\![(c_1,\ p_1) \rightarrow* (SKIP,\ p_2);\ (c_2,\ p_2) \rightarrow* (SKIP,\ p_3)]\!] \Longrightarrow$
$(c_1;;\ c_2,\ p_1) \rightarrow* (SKIP,\ p_3)$
⟨*proof*⟩

lemma *big-to-small*:
$cf \Rightarrow p \Longrightarrow cf \rightarrow* (SKIP,\ p)$
⟨*proof*⟩

lemma *small1-big-continue*:
$[\![cf \rightarrow cf';\ cf' \Rightarrow p]\!] \Longrightarrow cf \Rightarrow p$
⟨*proof*⟩

lemma *small-to-big*:
$cf \rightarrow* (SKIP,\ p) \Longrightarrow cf \Rightarrow p$
⟨*proof*⟩

lemma *big-iff-small*:

cf ⇒ *p* = *cf* →∗ (*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* × *nat*
type-synonym *config* = *state set* × *vname set*
type-synonym *scope* = *config set* × *bool*
type-synonym *state-upd* = *vname* × *val option*

definition *eq-streams* ::
stream ⇒ *stream* ⇒ *inputs* ⇒ *inputs* ⇒ *tag set* ⇒ *bool*

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

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

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

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

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

primrec *avars* :: *aexp* ⇒ *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* ⇒ *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-upd* :: *flow* ⇒ *vname set* ⇒ *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-aux* :: *com list* ⇒ *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* × *stage*) *list* ⇒ *flow* **where**
flow cfs = *flow-aux* (*map fst cfs*)

function *in-flow* :: *flow* ⇒ *inputs* ⇒ *stream* ⇒ *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*
n = *length* [*p*←*vs*. *fst p* = *x*] + *length* [*c*←*cs*. *c* = *IN x*]
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 [] - - = []

⟨*proof*⟩

termination ⟨*proof*⟩

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

⟨*proof*⟩

termination ⟨*proof*⟩

function *out-flow* :: *flow* ⇒ *inputs* ⇒ *state* ⇒ *stream* ⇒ *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*
in out-flow cs vs s f @ [(*x*, *t x*)]) |
out-flow (*cs* @ [⟨-⟩]) *vs s f* = *out-flow cs vs s f* |
out-flow [] - - - = []

⟨*proof*⟩

termination ⟨*proof*⟩

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* (⊆ *state*) ⟹ *interf s* = *interf t*

context *noninterf*
begin

As in my previous paper [10], function *sources* is defined along with an auxiliary function *sources-aux* 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 @ $[c_n]$, ..., cs_1 @ $[c_1]$ of *cs* and a sequence of variables y_{n+1}, ..., y_1, where $n \geq 1$, $cs_{n+1} = cs$, $y_{n+1} = x$, 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 ⇝ *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 ⇝ *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 ⇒ inputs ⇒ state ⇒ stream ⇒ vname ⇒ vname set and
  sources-aux :: flow ⇒ inputs ⇒ state ⇒ stream ⇒ vname ⇒ vname set
where

sources (cs @ [c]) vs s f x = (case c of
  z ::= a ⇒ if z = x
    then sources-aux cs vs s f x ∪ ⋃ {sources cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ avars a}
    else sources cs vs s f x |
  IN z ⇒ if z = x
    then sources-aux cs vs s f x
    else sources cs vs s f x |
  ⟨X⟩ ⇒
    sources cs vs s f x ∪ ⋃ {sources cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ X} |
  - ⇒
    sources cs vs s f x) |

sources [] - - - x = {x} |

sources-aux (cs @ [c]) vs s f x = (case c of
  ⟨X⟩ ⇒
    sources-aux cs vs s f x ∪ ⋃ {sources cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ X} |
  - ⇒
    sources-aux cs vs s f x) |

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

⟨proof⟩

termination ⟨proof⟩

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* ⇒ *inputs* ⇒ *state* ⇒ *stream* ⇒ *vname* ⇒ *vname set*
where

sources-out (*cs* @ [*c*]) *vs s f x* = (*case c of*
OUT z ⇒
sources-out cs vs s f x ∪ (*if z* = *x then sources cs vs s f x else* {}) |
⟨*X*⟩ ⇒
sources-out cs vs s f x ∪ ⋃ {*sources cs vs s f y* | *y*.
run-flow cs vs s f: *dom y* ⇝ *dom x* ∧ *y* ∈ *X*} |
- ⇒
sources-out cs vs s f x) |

sources-out [] - - - - = {}

⟨*proof*⟩
termination ⟨*proof*⟩

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 ⇒ inputs ⇒ state ⇒ stream ⇒ vname ⇒ tag set and
  tags-aux :: flow ⇒ inputs ⇒ state ⇒ stream ⇒ vname ⇒ tag set
where

tags (cs @ [c]) vs s f x = (case c of
  z ::= a ⇒ if z = x
    then tags-aux cs vs s f x ∪ ⋃ {tags cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ avars a}
    else tags cs vs s f x |
  IN z ⇒ if z = x
    then insert (x, length [c←cs. c = IN x]) (tags-aux cs vs s f x)
    else tags cs vs s f x |
  ⟨X⟩ ⇒
    tags cs vs s f x ∪ ⋃ {tags cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ X} |
  - ⇒
    tags cs vs s f x) |

tags [] - - - - = {} |

tags-aux (cs @ [c]) vs s f x = (case c of
  ⟨X⟩ ⇒
    tags-aux cs vs s f x ∪ ⋃ {tags cs vs s f y | y.
      run-flow cs vs s f: dom y ⇝ dom x ∧ y ∈ X} |
  - ⇒
    tags-aux cs vs s f x) |

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

⟨proof⟩
```

termination ⟨*proof*⟩

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* ⇒ *inputs* ⇒ *state* ⇒ *stream* ⇒ *vname* ⇒ *tag set*
where

tags-out (*cs* @ [*c*]) *vs s f x* = (*case c of*
OUT z ⇒
tags-out cs vs s f x ∪ (*if z* = *x then tags cs vs s f x else* {}) |
⟨*X*⟩ ⇒
tags-out cs vs s f x ∪ ⋃ {*tags cs vs s f y* | *y*.
run-flow cs vs s f: *dom y* ⇝ *dom x* ∧ *y* ∈ *X*} |
- ⇒
tags-out cs vs s f x) |

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

⟨*proof*⟩
termination ⟨*proof*⟩

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

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

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

definition *correct* :: *com* ⇒ *state set* ⇒ *vname set* ⇒ *bool* **where**
correct $c\ A\ X \equiv$
$\forall s \in Univ\ A\ (\subseteq state \cap X).\ \forall c_1\ c_2\ s_1\ s_2\ f\ vs\ vs_1\ vs_2\ ws\ ws_1\ ws_2\ cfs.$
$(c, s, f, vs, ws) \rightarrow* (c_1, s_1, f, vs_1, ws_1) \wedge$
$(c_1, s_1, f, vs_1, ws_1) \rightarrow*\{cfs\}\ (c_2, s_2, f, vs_2, ws_2) \longrightarrow$
$ok\text{-}flow\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs)$

abbreviation *noninterf-set* :: *state set* ⇒ *vname set* ⇒ *vname set* ⇒ *bool*
$((-{:}\ -\ \rightsquigarrow|\ -)\ [51, 51, 51]\ 50)$ **where**
$A{:}\ X \rightsquigarrow|\ Y \equiv \forall y \in Y.\ \exists s \in A.\ \exists x \in X.\ \neg\ s{:}\ dom\ x \rightsquigarrow dom\ y$

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

```
    S ≠ {} ⟶
    f = f' (⊆ vs1, vs1', ⋃ {tags-aux cs vs1 s1 f x | x. x ∈ S}) ⟶
      (c1, t1, f', vs1', ws1') →* (c2', t2, f', vs2', ws2') ∧
      (c2 = SKIP) = (c2' = SKIP) ∧
      map fst [p←drop (length vs1) vs2. fst p ∈ S] =
        map fst [p←drop (length vs1') vs2'. fst p ∈ S]

abbreviation ok-flow-aux-2 where
ok-flow-aux-2 s1 s2 t1 t2 f f' vs1 vs1' cs ≡
  ∀ S ⊆ {x. s1 = t1 (⊆ sources cs vs1 s1 f x)}.
    S ≠ {} ⟶
    f = f' (⊆ vs1, vs1', ⋃ {tags cs vs1 s1 f x | x. x ∈ S}) ⟶
      s2 = t2 (⊆ S)

abbreviation ok-flow-aux-3 where
ok-flow-aux-3 s1 t1 f f' vs1 vs1' ws1 ws1' ws2 ws2' cs ≡
  ∀ S ⊆ {x. s1 = t1 (⊆ sources-out cs vs1 s1 f x)}.
    S ≠ {} ⟶
    f = f' (⊆ vs1, vs1', ⋃ {tags-out cs vs1 s1 f x | x. x ∈ S}) ⟶
      [p←drop (length ws1) ws2. fst p ∈ S] =
        [p←drop (length ws1') ws2'. fst p ∈ S]

abbreviation ok-flow-aux :: config set ⇒ com ⇒ com ⇒ state ⇒ state ⇒
  stream ⇒ inputs ⇒ inputs ⇒ outputs ⇒ outputs ⇒ flow ⇒ bool
where
ok-flow-aux U c1 c2 s1 s2 f vs1 vs2 ws1 ws2 cs ≡
  (∀ t1 f' vs1' ws1'. ∃ c2' t2 vs2' ws2'.
    ok-flow-aux-1 c1 c2 c2' s1 t1 t2 f f' vs1 vs1' vs2 vs2' ws1' ws2' cs ∧
    ok-flow-aux-2 s1 s2 t1 t2 f f' vs1 vs1' cs ∧
    ok-flow-aux-3 s1 t1 f f' vs1 vs1' ws1 ws1' ws2 ws2' cs) ∧
  (∀ Y. (∃ (A, X) ∈ U. A: X ⇝| Y) ⟶ no-upd cs Y)
```

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* × *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, *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, *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* ⇒ *bool option* ((⊢ -) [*51*] *55*) **where**

⊢ *Bc v* = *Some v* |

⊢ *Not b* = (*case* ⊢ *b of*
Some v ⇒ *Some* (¬ *v*) | - ⇒ *None*) |

⊢ *And* b_1 b_2 = (*case* (⊢ b_1, ⊢ b_2) *of*
(*Some* v_1, *Some* v_2) ⇒ *Some* (v_1 ∧ v_2) | - ⇒ *None*) |

⊢ *Less* a_1 a_2 = (*if avars* a_1 ∪ *avars* a_2 = {}
then Some (*aval* a_1 (λ*x*. *0*) < *aval* a_2 (λ*x*. *0*)) *else None*)

inductive-set *ctyping1-merge-aux* :: *state-upd list set* ⇒
state-upd list set ⇒ (*state-upd list* × *bool*) *list set*
(**infix** ⨆ *55*) **for** *A* **and** *B* **where**

xs ∈ *A* ⟹ [(*xs*, *True*)] ∈ *A* ⨆ *B* |

ys ∈ *B* ⟹ [(*ys*, *False*)] ∈ *A* ⨆ *B* |

⟦*ws* ∈ *A* ⨆ *B*; ¬ *snd* (*hd ws*); *xs* ∈ *A*; (*xs*, *True*) ∉ *set ws*⟧ ⟹
(*xs*, *True*) # *ws* ∈ *A* ⨆ *B* |

⟦*ws* ∈ *A* ⨆ *B*; *snd* (*hd ws*); *ys* ∈ *B*; (*ys*, *False*) ∉ *set ws*⟧ ⟹
(*ys*, *False*) # *ws* ∈ *A* ⨆ *B*

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

definition *ctyping1-append* ::
state-upd list set ⇒ *state-upd list set* ⇒ *state-upd list set*
(**infixl** @ *55*) **where**
A @ $B \equiv \{xs$ @ $ys \mid xs\ ys.\ xs \in A \wedge ys \in B\}$

definition *ctyping1-merge* ::
state-upd list set ⇒ *state-upd list set* ⇒ *state-upd list set*
(**infixl** ⊔ *55*) **where**
$A \sqcup B \equiv \{concat\ (map\ fst\ ws) \mid ws.\ ws \in A \bigsqcup B\}$

definition *ctyping1-merge-append* ::
state-upd list set ⇒ *state-upd list set* ⇒ *state-upd list set*
(**infixl** $\sqcup_@$ *55*) **where**
$A \sqcup_@ B \equiv (if\ card\ B = Suc\ 0\ then\ A\ else\ A \sqcup B)$ @ B

primrec *ctyping1-aux* :: *com* ⇒ *state-upd list set*
((⊢ -) [*51*] *60*) **where**

⊢ *SKIP* = {[]} |

⊢ $x ::= a$ = (*if* $x \in state$
then {[(x, *if avars* $a = \{\}$ *then Some* (*aval* a ($\lambda x.\ 0$)) *else None*)]}
else {[]}) |

⊢ *IN* x = (*if* $x \in state$ *then* {[(x, *None*)]} *else* {[]}) |

⊢ *OUT* x = {[]} |

⊢ c_1;; c_2 = ⊢ c_1 $\sqcup_@$ ⊢ c_2 |

⊢ c_1 *OR* c_2 = ⊢ c_1 ⊔ ⊢ c_2 |

⊢ *IF* b *THEN* c_1 *ELSE* c_2 = (*let* f = ⊢ b *in*
(*if* $f \in$ {*Some True*, *None*} *then* ⊢ c_1 *else* {}) ⊔
(*if* $f \in$ {*Some False*, *None*} *then* ⊢ c_2 *else* {})) |

⊢ *WHILE* b *DO* c = (*let* f = ⊢ b *in*
(*if* $f \in$ {*Some False*, *None*} *then* {[]} *else* {}) ∪
(*if* $f \in$ {*Some True*, *None*} *then* ⊢ c *else* {}))

definition *ctyping1* :: *com* ⇒ *state set* ⇒ *vname set* ⇒ *config*
((⊢ - ′(⊆ -, -′)) [*51*] *55*) **where**
⊢ c (⊆ A, X) ≡ *let* F = {$\lambda x.$ [y←$ys.\ fst\ y = x$] | $ys.\ ys \in$ ⊢ c} *in*
({$\lambda x.$ *if* $f\ x$ = []
then $s\ x$ *else case snd* (*last* ($f\ x$)) *of None* ⇒ $t\ x$ | *Some* i ⇒ i |

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

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* ⇒ *state set* ⇒ *vname set* ⇒ *state set option*
((⊨ - ′(⊆ -, -′)) [*51*] *55*) **where**

⊨ *Bc v* (⊆ *A*, -) = *Some* (*if v then A else* {}) |

⊨ *Not b* (⊆ *A*, *X*) = (*case* ⊨ *b* (⊆ *A*, *X*) *of*
Some B ⇒ *Some* (*A* − *B*) | - ⇒ *None*) |

⊨ *And* $b_1\ b_2$ (⊆ *A*, *X*) = (*case* (⊨ b_1 (⊆ *A*, *X*), ⊨ b_2 (⊆ *A*, *X*)) *of*
(*Some* B_1, *Some* B_2) ⇒ *Some* ($B_1 \cap B_2$) | - ⇒ *None*) |

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

definition *btyping2* ::

```
 bexp ⇒ state set ⇒ vname set ⇒ state set × state set
  ((⊨ - ′(⊆ -, -′)) [51] 55) where
⊨ b (⊆ A, X) ≡ case ⊭ b (⊆ A, X) of
  Some A′ ⇒ (A′, A − A′) | - ⇒ (A, A)


abbreviation interf-set :: state set ⇒ vname set ⇒ vname set ⇒ bool
  ((-: - ⇝ -) [51, 51, 51] 50) where
A: X ⇝ Y ≡ ∀ s ∈ A. ∀ x ∈ X. ∀ y ∈ Y. s: dom x ⇝ dom y

abbreviation atyping :: bool ⇒ aexp ⇒ vname set ⇒ bool
  ((- ⊨ - ′(⊆ -′)) [51, 51] 50) where
v ⊨ a (⊆ X) ≡ avars a = {} ∨ avars a ⊆ state ∩ X ∧ v

definition univ-states-if :: state set ⇒ vname set ⇒ state set
  ((Univ? - -) [51, 75] 75) where
Univ? A X ≡ if state ⊆ X then A else Univ A (⊆ {})


fun ctyping2 :: scope ⇒ com ⇒ state set ⇒ vname set ⇒ config option
  ((- ⊨ - ′(⊆ -, -′)) [51, 51] 55) where

- ⊨ SKIP (⊆ A, X) = Some (A, Univ?? A X) |

(U, v) ⊨ x ::= a (⊆ A, X) =
 (if ∀ (B, Y) ∈ insert (Univ? A X, avars a) U. B: Y ⇝ {x}
  then Some (if x ∈ state ∧ A ≠ {}
    then if v ⊨ a (⊆ X)
      then ({s(x := aval a s) | s. s ∈ A}, insert x X) else (A, X − {x})
    else (A, Univ?? A X))
  else None) |

(U, v) ⊨ IN x (⊆ A, X) =
 (if ∀ (B, Y) ∈ U. B: Y ⇝ {x}
  then Some (if x ∈ state ∧ A ≠ {}
    then (A, X − {x}) else (A, Univ?? A X))
  else None) |

(U, v) ⊨ OUT x (⊆ A, X) =
 (if ∀ (B, Y) ∈ U. B: Y ⇝ {x}
  then Some (A, Univ?? A X)
  else None) |

(U, v) ⊨ c1;; c2 (⊆ A, X) =
 (case (U, v) ⊨ c1 (⊆ A, X) of
    Some (B, Y) ⇒ (U, v) ⊨ c2 (⊆ B, Y) | - ⇒ None) |

(U, v) ⊨ c1 OR c2 (⊆ A, X) =
 (case ((U, v) ⊨ c1 (⊆ A, X), (U, v) ⊨ c2 (⊆ A, X)) of
```

```
    (Some (C1, Y1), Some (C2, Y2)) ⇒ Some (C1 ∪ C2, Y1 ∩ Y2) |
    - ⇒ None) |

(U, v) ⊨ IF b THEN c1 ELSE c2 (⊆ A, X) =
 (case (insert (Univ? A X, bvars b) U, ⊨ b (⊆ A, X)) of (U', B1, B2) ⇒
    case ((U', v) ⊨ c1 (⊆ B1, X), (U', v) ⊨ c2 (⊆ B2, X)) of
      (Some (C1, Y1), Some (C2, Y2)) ⇒ Some (C1 ∪ C2, Y1 ∩ Y2) |
      - ⇒ None) |

(U, v) ⊨ WHILE b DO c (⊆ A, X) = (case ⊨ b (⊆ A, X) of (B1, B2) ⇒
  case ⊢ c (⊆ B1, X) of (C, Y) ⇒ case ⊨ b (⊆ C, Y) of (B1', B2') ⇒
    if ∀(B, W) ∈ insert (Univ? A X ∪ Univ? C Y, bvars b) U. B: W ⇝ UNIV
    then case (({}, False) ⊨ c (⊆ B1, X), ({}, False) ⊨ c (⊆ B1', Y)) of
      (Some -, Some -) ⇒ Some (B2 ∪ B2', Univ?? B2 X ∩ Y) |
      - ⇒ None
    else None)

end

end
```

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 *vname 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 ≡
  if [y←ys'. fst y = x] = []
  then if [y←ys. fst y = x] = []
    then s x
    else case snd (last [y←ys. fst y = x]) of None ⇒ t x | Some i ⇒ i
  else case snd (last [y←ys'. fst y = x]) of None ⇒ t' x | Some i ⇒ i
```

abbreviation *ctyping1-idem-rhs* **where**
ctyping1-idem-rhs f s t x ≡
if f x = []
then s x
else case snd (*last* (*f x*)) *of None* ⇒ *t x* | *Some i* ⇒ *i*

abbreviation *ctyping1-idem-pred* **where**
ctyping1-idem-pred s t t′ ys ys′ A (*S* :: *state-upd list set*) ≡ ∃*f s′*.
(∃ *t″*. *ctyping1-idem-lhs s t t′ ys ys′* = *ctyping1-idem-rhs f s′ t″*) ∧
(∀ *x*. (*f x* = [] ⟷ [*y*←*ys* @ *ys′*. *fst y* = *x*] = []) ∧
(*f x* ≠ [] ⟶ *last* (*f x*) = *last* [*y*←*ys* @ *ys′*. *fst y* = *x*])) ∧
(∃ *ys″*. *f* = (λ*x*. [*y*←*ys″*. *fst y* = *x*]) ∧ *ys″* ∈ *S*) ∧ *s′* ∈ *A*

lemma *ctyping1-merge-aux-no-nil*:
ws ∈ *A* ⨆ *B* ⟹ *ws* ≠ []
⟨*proof*⟩

lemma *ctyping1-merge-aux-empty-lhs*:
{} ⨆ *B* = {[(*ys*, *False*)] | *ys*. *ys* ∈ *B*}
⟨*proof*⟩

lemma *ctyping1-merge-aux-empty-rhs*:
A ⨆ {} = {[(*xs*, *True*)] | *xs*. *xs* ∈ *A*}
⟨*proof*⟩

lemma *ctyping1-merge-empty-lhs*:
{} ⊔ *B* = *B*
⟨*proof*⟩

lemma *ctyping1-merge-empty-rhs*:
A ⊔ {} = *A*
⟨*proof*⟩

lemma *ctyping1-aux-nonempty*:
⊢ *c* ≠ {}
⟨*proof*⟩

lemma *ctyping1-merge-idem-fst*:
assumes
A: ⋀*ys ys′*. *ys* ∈ ⊢ *c*$_1$ ⟹ *ys′* ∈ ⊢ *c*$_1$ ⟹
ctyping1-idem-pred s t t′ ys ys′ A (⊢ *c*$_1$) **and**
B: ⋀*ys ys′*. *ys* ∈ ⊢ *c*$_2$ ⟹ *ys′* ∈ ⊢ *c*$_2$ ⟹
ctyping1-idem-pred s t t′ ys ys′ A (⊢ *c*$_2$) **and**
C: *s* ∈ *A* **and**
D: *ys* ∈ ⊢ *c*$_1$ ⊔ ⊢ *c*$_2$ **and**
E: *ys′* ∈ ⊢ *c*$_1$ ⊔ ⊢ *c*$_2$

shows *ctyping1-idem-pred s t t′ ys ys′ A* (⊢ c_1 ⊔ ⊢ c_2)
⟨*proof*⟩

lemma *ctyping1-merge-append-idem-fst*:
assumes
A: ⋀*ys ys′*. *ys* ∈ ⊢ c_1 ⟹ *ys′* ∈ ⊢ c_1 ⟹
ctyping1-idem-pred s t t′ ys ys′ A (⊢ c_1) **and**
B: ⋀*ys ys′*. *ys* ∈ ⊢ c_2 ⟹ *ys′* ∈ ⊢ c_2 ⟹
ctyping1-idem-pred s t t′ ys ys′ A (⊢ c_2) **and**
C: *s* ∈ *A* **and**
D: *ys* ∈ ⊢ c_1 ⊔@ ⊢ c_2 **and**
E: *ys′* ∈ ⊢ c_1 ⊔@ ⊢ c_2
shows *ctyping1-idem-pred s t t′ ys ys′ A* (⊢ c_1 ⊔@ ⊢ c_2)
⟨*proof*⟩

lemma *ctyping1-aux-idem-fst*:
⟦*s* ∈ *A*; *ys* ∈ ⊢ *c*; *ys′* ∈ ⊢ *c*⟧ ⟹
ctyping1-idem-pred s t t′ ys ys′ A (⊢ *c*)
⟨*proof*⟩

lemma *ctyping1-idem-fst-1*:
⟦*s* ∈ *A*; *ys* ∈ ⊢ *c*; *ys′* ∈ ⊢ *c*⟧ ⟹ ∃ *f s′*.
(∃ *t″*. *ctyping1-idem-lhs s t t′ ys ys′* = *ctyping1-idem-rhs f s′ t″*) ∧
(∃ *ys″*. *f* = (λ*x*. [*y*←*ys″*. *fst y* = *x*]) ∧ *ys″* ∈ ⊢ *c*) ∧ *s′* ∈ *A*
⟨*proof*⟩

lemma *ctyping1-idem-fst-2*:
⟦*s* ∈ *A*; *ys* ∈ ⊢ *c*⟧ ⟹ ∃ *f s′*.
(∃ *t′*.
(λ*x*. *if* [*y*←*ys*. *fst y* = *x*] = []
then s x
else case snd (*last* [*y*←*ys*. *fst y* = *x*]) *of None* ⇒ *t x* | *Some i* ⇒ *i*) =
(λ*x*. *if f x* = []
then s′ x
else case snd (*last* (*f x*)) *of None* ⇒ *t′ x* | *Some i* ⇒ *i*)) ∧
(∃ *ys′*. *f* = (λ*x*. [*y*←*ys′*. *fst y* = *x*]) ∧ *ys′* ∈ ⊢ *c*) ∧
(∃ *f′ s″*.
(∃ *t″*. *s′* = (λ*x*. *if f′ x* = []
then s″ x
else case snd (*last* (*f′ x*)) *of None* ⇒ *t″ x* | *Some i* ⇒ *i*)) ∧
(∃ *ys″*. *f′* = (λ*x*. [*y*←*ys″*. *fst y* = *x*]) ∧ *ys″* ∈ ⊢ *c*) ∧ *s″* ∈ *A*)
(**is** ⟦-; -⟧ ⟹ ∃ - -. (∃ -. *?f* = -) ∧ -)
⟨*proof*⟩

lemma *ctyping1-idem-fst*:
⊢ *c* (⊆ *A*, *X*) = (*B*, *Y*) ⟹ *case* ⊢ *c* (⊆ *B*, *Y*) *of* (*B′*, *Y′*) ⇒ *B′* = *B*
⟨*proof*⟩

lemma *ctyping1-idem-snd-1*:
assumes
A: $A \neq \{\}$ **and**
B: $\forall\, r\ f\ s.$
$(\forall\, t.\ r \neq (\lambda x.\ \mathit{if}\ f\ x = []\ \mathit{then}\ s\ x\ \mathit{else}\ \mathit{case}\ \mathit{snd}\ (\mathit{last}\ (f\ x))\ \mathit{of}$
$\mathit{None} \Rightarrow t\ x \mid \mathit{Some}\ i \Rightarrow i)) \vee$
$(\forall\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \longrightarrow ys \notin \vdash c) \vee s \notin A$
(**is** $\forall\, r\ f\ s.\ (\forall\, t.\ r \neq ?r\ f\ s\ t) \vee$ -)
shows $\mathit{UNIV} = S$
⟨*proof*⟩

lemma *ctyping1-idem-snd-2*:
$\{x.\ \forall f.$
$(f\ x = [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$(\forall f.$
$(f\ x = [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$x \in X) \wedge$
$(f\ x \neq [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$(\exists\, i.\ \mathit{snd}\ (\mathit{last}\ (f\ x)) = \mathit{Some}\ i)))) \wedge$
$(f\ x \neq [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$(\exists\, i.\ \mathit{snd}\ (\mathit{last}\ (f\ x)) = \mathit{Some}\ i))\} =$
$\{x.\ \forall f.$
$(f\ x = [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$x \in X) \wedge$
$(f\ x \neq [] \longrightarrow (\exists\, ys.\ f = (\lambda x.\ [y{\leftarrow}ys.\ \mathit{fst}\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
$(\exists\, i.\ \mathit{snd}\ (\mathit{last}\ (f\ x)) = \mathit{Some}\ i))\}$
⟨*proof*⟩

lemma *ctyping1-idem-snd*:
$\vdash c\ (\subseteq A,\ X) = (B,\ Y) \Longrightarrow \mathit{case} \vdash c\ (\subseteq B,\ Y)\ \mathit{of}\ (B',\ Y') \Rightarrow Y' = Y$
⟨*proof*⟩

lemma *ctyping1-idem*:
$\vdash c\ (\subseteq A,\ X) = (B,\ Y) \Longrightarrow \vdash c\ (\subseteq B,\ Y) = (B,\ Y)$
⟨*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*:
$s = t\ (\subseteq avars\ a) \Longrightarrow aval\ a\ s = aval\ a\ t$
⟨*proof*⟩

4.2 Local context proofs

context *noninterf*
begin

lemma *interf-set-mono*:
$[\![A' \subseteq A;\ X \subseteq X';\ \forall (B',\ Y') \in U'.\ \exists (B,\ Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y;$
$\forall (B,\ Y) \in insert\ (Univ?\ A\ X,\ Z)\ U.\ B{:}\ Y \rightsquigarrow W]\!] \Longrightarrow$
$\forall (B,\ Y) \in insert\ (Univ?\ A'\ X',\ Z)\ U'.\ B{:}\ Y \rightsquigarrow W$
⟨*proof*⟩

lemma *btyping1-btyping2-aux-1* [*elim*]:
assumes
$A{:}\ avars\ a_1 = \{\}$ **and**
$B{:}\ avars\ a_2 = \{\}$ **and**
$C{:}\ aval\ a_1\ (\lambda x.\ 0) < aval\ a_2\ (\lambda x.\ 0)$
shows $aval\ a_1\ s < aval\ a_2\ s$
⟨*proof*⟩

lemma *btyping1-btyping2-aux-2* [*elim*]:
assumes
$A{:}\ avars\ a_1 = \{\}$ **and**
$B{:}\ avars\ a_2 = \{\}$ **and**
$C{:}\ \neg\ aval\ a_1\ (\lambda x.\ 0) < aval\ a_2\ (\lambda x.\ 0)$ **and**
$D{:}\ aval\ a_1\ s < aval\ a_2\ s$
shows *False*
⟨*proof*⟩

lemma *btyping1-btyping2-aux*:
$\vdash b = Some\ v \Longrightarrow \models b\ (\subseteq A,\ X) = Some\ (if\ v\ then\ A\ else\ \{\})$
⟨*proof*⟩

lemma *btyping1-btyping2*:
$\vdash b = Some\ v \Longrightarrow \models b\ (\subseteq A,\ X) = (if\ v\ then\ (A,\ \{\})\ else\ (\{\},\ A))$
⟨*proof*⟩

lemma *btyping2-aux-subset*:
$\models b\ (\subseteq A,\ X) = Some\ A' \Longrightarrow A' = \{s.\ s \in A \wedge bval\ b\ s\}$
⟨*proof*⟩

lemma *btyping2-aux-diff*:
$[\![\models b\ (\subseteq A,\ X) = Some\ B;\ \models b\ (\subseteq A',\ X') = Some\ B';\ A' \subseteq A;\ B' \subseteq B]\!] \Longrightarrow$
$A' - B' \subseteq A - B$
⟨*proof*⟩

lemma *btyping2-aux-mono*:
$[\![\models b\ (\subseteq A,\ X) = Some\ B;\ A' \subseteq A;\ X \subseteq X']\!] \Longrightarrow$
$\exists B'.\ \models b\ (\subseteq A',\ X') = Some\ B' \wedge B' \subseteq B$
⟨*proof*⟩

lemma *btyping2-mono*:
$[\![\models b\ (\subseteq A,\ X) = (B_1,\ B_2);\ \models b\ (\subseteq A',\ X') = (B_1{}',\ B_2{}');\ A' \subseteq A;\ X \subseteq X']\!] \Longrightarrow$
$B_1{}' \subseteq B_1 \wedge B_2{}' \subseteq B_2$
⟨*proof*⟩

lemma *btyping2-un-eq*:
$\models b\ (\subseteq A,\ X) = (B_1,\ B_2) \Longrightarrow B_1 \cup B_2 = A$
⟨*proof*⟩

lemma *btyping2-aux-eq*:
$[\![\models b\ (\subseteq A,\ X) = Some\ A';\ s = t\ (\subseteq state \cap X)]\!] \Longrightarrow bval\ b\ s = bval\ b\ t$
⟨*proof*⟩

lemma *ctyping1-mono-fst*:
$[\![\vdash c\ (\subseteq A,\ X) = (B,\ Y);\ \vdash c\ (\subseteq A',\ X') = (B',\ Y');\ A' \subseteq A]\!] \Longrightarrow$
$B' \subseteq B$
⟨*proof*⟩

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 $B' \subseteq B \wedge Y \subseteq Y'$

⟨*proof*⟩

lemma *ctyping2-mono-skip* [*elim*!]:
⟦(*U*, *False*) ⊨ *SKIP* (⊆ *A*, *X*) = *Some* (*C*, *Z*); *A'* ⊆ *A*; *X* ⊆ *X'*⟧ ⟹
∃ *C' Z'*. (*U'*, *False*) ⊨ *SKIP* (⊆ *A'*, *X'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'*
⟨*proof*⟩

lemma *ctyping2-mono-assign* [*elim*!]:
⟦(*U*, *False*) ⊨ *x* ::= *a* (⊆ *A*, *X*) = *Some* (*C*, *Z*); *A'* ⊆ *A*; *X* ⊆ *X'*;
∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y*⟧ ⟹
∃ *C' Z'*. (*U'*, *False*) ⊨ *x* ::= *a* (⊆ *A'*, *X'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'*
⟨*proof*⟩

lemma *ctyping2-mono-input* [*elim*!]:
⟦(*U*, *False*) ⊨ *IN x* (⊆ *A*, *X*) = *Some* (*C*, *Z*); *A'* ⊆ *A*; *X* ⊆ *X'*;
∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y*⟧ ⟹
∃ *C' Z'*. (*U'*, *False*) ⊨ *IN x* (⊆ *A'*, *X'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'*
⟨*proof*⟩

lemma *ctyping2-mono-output* [*elim*!]:
⟦(*U*, *False*) ⊨ *OUT x* (⊆ *A*, *X*) = *Some* (*C*, *Z*); *A'* ⊆ *A*; *X* ⊆ *X'*;
∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y*⟧ ⟹
∃ *C' Z'*. (*U'*, *False*) ⊨ *OUT x* (⊆ *A'*, *X'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'*
⟨*proof*⟩

lemma *ctyping2-mono-seq*:
assumes
A: ⋀*A' B X' Y U'*.
(*U*, *False*) ⊨ *c*1 (⊆ *A*, *X*) = *Some* (*B*, *Y*) ⟹ *A'* ⊆ *A* ⟹ *X* ⊆ *X'* ⟹
∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y* ⟹
∃ *B' Y'*. (*U'*, *False*) ⊨ *c*1 (⊆ *A'*, *X'*) = *Some* (*B'*, *Y'*) ∧
B' ⊆ *B* ∧ *Y* ⊆ *Y'* **and**
B: ⋀*p B Y B' C Y' Z U'*.
(*U*, *False*) ⊨ *c*1 (⊆ *A*, *X*) = *Some p* ⟹ (*B*, *Y*) = *p* ⟹
(*U*, *False*) ⊨ *c*2 (⊆ *B*, *Y*) = *Some* (*C*, *Z*) ⟹ *B'* ⊆ *B* ⟹ *Y* ⊆ *Y'* ⟹
∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y* ⟹
∃ *C' Z'*. (*U'*, *False*) ⊨ *c*2 (⊆ *B'*, *Y'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'* **and**
C: (*U*, *False*) ⊨ *c*1;; *c*2 (⊆ *A*, *X*) = *Some* (*C*, *Z*) **and**
D: *A'* ⊆ *A* **and**
E: *X* ⊆ *X'* **and**
F: ∀(*B'*, *Y'*) ∈ *U'*. ∃(*B*, *Y*) ∈ *U*. *B'* ⊆ *B* ∧ *Y'* ⊆ *Y*
shows ∃ *C' Z'*. (*U'*, *False*) ⊨ *c*1;; *c*2 (⊆ *A'*, *X'*) = *Some* (*C'*, *Z'*) ∧
C' ⊆ *C* ∧ *Z* ⊆ *Z'*

⟨*proof*⟩

lemma *ctyping2-mono-or*:
assumes
A: $\bigwedge A'\ C_1\ X'\ Y_1\ U'.$
$(U,\ False) \models c_1\ (\subseteq A,\ X) = Some\ (C_1,\ Y_1) \Longrightarrow A' \subseteq A \Longrightarrow X \subseteq X' \Longrightarrow$
$\forall (B',\ Y') \in U'.\ \exists (B,\ Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y \Longrightarrow$
$\exists\, C_1'\ Y_1'.\ (U',\ False) \models c_1\ (\subseteq A',\ X') = Some\ (C_1',\ Y_1') \wedge$
$C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1'$ **and**
B: $\bigwedge A'\ C_2\ X'\ Y_2\ U'.$
$(U,\ False) \models c_2\ (\subseteq A,\ X) = Some\ (C_2,\ Y_2) \Longrightarrow A' \subseteq A \Longrightarrow X \subseteq X' \Longrightarrow$
$\forall (B',\ Y') \in U'.\ \exists (B,\ Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y \Longrightarrow$
$\exists\, C_2'\ Y_2'.\ (U',\ False) \models c_2\ (\subseteq A',\ X') = Some\ (C_2',\ Y_2') \wedge$
$C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2'$ **and**
C: $(U,\ False) \models c_1\ OR\ c_2\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
D: $A' \subseteq A$ **and**
E: $X \subseteq X'$ **and**
F: $\forall (B',\ Y') \in U'.\ \exists (B,\ Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y$
shows $\exists\, C'\ Y'.\ (U',\ False) \models c_1\ OR\ c_2\ (\subseteq A',\ X') = Some\ (C',\ Y') \wedge$
$C' \subseteq C \wedge Y \subseteq Y'$
⟨*proof*⟩

lemma *ctyping2-mono-if*:
assumes
A: $\bigwedge W\ p\ B_1\ B_2\ B_1'\ C_1\ X'\ Y_1\ W'.\ (W,\ p) =$
$(insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow (B_1,\ B_2) = p \Longrightarrow$
$(W,\ False) \models c_1\ (\subseteq B_1,\ X) = Some\ (C_1,\ Y_1) \Longrightarrow B_1' \subseteq B_1 \Longrightarrow$
$X \subseteq X' \Longrightarrow \forall (B',\ Y') \in W'.\ \exists (B,\ Y) \in W.\ B' \subseteq B \wedge Y' \subseteq Y \Longrightarrow$
$\exists\, C_1'\ Y_1'.\ (W',\ False) \models c_1\ (\subseteq B_1',\ X') = Some\ (C_1',\ Y_1') \wedge$
$C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1'$ **and**
B: $\bigwedge W\ p\ B_1\ B_2\ B_2'\ C_2\ X'\ Y_2\ W'.\ (W,\ p) =$
$(insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow (B_1,\ B_2) = p \Longrightarrow$
$(W,\ False) \models c_2\ (\subseteq B_2,\ X) = Some\ (C_2,\ Y_2) \Longrightarrow B_2' \subseteq B_2 \Longrightarrow$
$X \subseteq X' \Longrightarrow \forall (B',\ Y') \in W'.\ \exists (B,\ Y) \in W.\ B' \subseteq B \wedge Y' \subseteq Y \Longrightarrow$
$\exists\, C_2'\ Y_2'.\ (W',\ False) \models c_2\ (\subseteq B_2',\ X') = Some\ (C_2',\ Y_2') \wedge$
$C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2'$ **and**
C: $(U,\ False) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
D: $A' \subseteq A$ **and**
E: $X \subseteq X'$ **and**
F: $\forall (B',\ Y') \in U'.\ \exists (B,\ Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y$
shows $\exists\, C'\ Y'.\ (U',\ False) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A',\ X') =$
$Some\ (C',\ Y') \wedge C' \subseteq C \wedge Y \subseteq Y'$
⟨*proof*⟩

lemma *ctyping2-mono-while*:
assumes
A: $\bigwedge B_1\ B_2\ C\ Y\ B_1'\ B_2'\ D_1\ E\ X'\ V\ U'.\ (B_1,\ B_2) = \models b\ (\subseteq A,\ X) \Longrightarrow$
$(C,\ Y) = \vdash c\ (\subseteq B_1,\ X) \Longrightarrow (B_1',\ B_2') = \models b\ (\subseteq C,\ Y) \Longrightarrow$
$\forall (B,\ W) \in insert\ (Univ?\ A\ X \cup Univ?\ C\ Y,\ bvars\ b)\ U.$

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

lemma *ctyping2-mono*:
$[\![(U, False) \models c\ (\subseteq A, X) = Some\ (C, Z);\ A' \subseteq A;\ X \subseteq X';$
$\forall (B', Y') \in U'.\ \exists (B, Y) \in U.\ B' \subseteq B \wedge Y' \subseteq Y]\!] \Longrightarrow$
$\exists C'\ Z'.\ (U', False) \models c\ (\subseteq A', X') = Some\ (C', Z') \wedge C' \subseteq C \wedge Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-assign* [*elim*!]:
assumes
A: $\vdash x ::= a\ (\subseteq A, X) = (C, Z)$ **and**
B: $(U, False) \models x ::= a\ (\subseteq A, X) = Some\ (C', Z')$
shows $C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-input* [*elim*!]:
assumes
A: $\vdash IN\ x\ (\subseteq A, X) = (C, Z)$ **and**
B: $(U, False) \models IN\ x\ (\subseteq A, X) = Some\ (C', Z')$
shows $C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-output* [*elim*!]:
$[\![\vdash OUT\ x\ (\subseteq A, X) = (C, Z);$
$(U, False) \models OUT\ x\ (\subseteq A, X) = Some\ (C', Z')]\!] \Longrightarrow$
$C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-seq*:

assumes
A: $\vdash c_1;;\ c_2\ (\subseteq A,\ X) = (C,\ Z)$ **and**
B: $(U,\ \mathit{False}) \models c_1;;\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Z')$ **and**
C: $\bigwedge B\ B'\ Y\ Y'.\ \vdash c_1\ (\subseteq A,\ X) = (B,\ Y) \Longrightarrow$
$(U,\ \mathit{False}) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ (B',\ Y') \Longrightarrow B' \subseteq B$ **and**
D: $\bigwedge p\ B'\ Y'\ D'\ C'\ W'\ Z'.$
$(U,\ \mathit{False}) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ p \Longrightarrow (B',\ Y') = p \Longrightarrow$
$\vdash c_2\ (\subseteq B',\ Y') = (D',\ W') \Longrightarrow$
$(U,\ \mathit{False}) \models c_2\ (\subseteq B',\ Y') = \mathit{Some}\ (C',\ Z') \Longrightarrow C' \subseteq D'$
shows $C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-or*:
assumes
A: $\vdash c_1\ \mathit{OR}\ c_2\ (\subseteq A,\ X) = (C,\ Y)$ **and**
B: $(U,\ \mathit{False}) \models c_1\ \mathit{OR}\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Y')$ **and**
C: $\bigwedge C\ C'\ Y\ Y'.\ \vdash c_1\ (\subseteq A,\ X) = (C,\ Y) \Longrightarrow$
$(U,\ \mathit{False}) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Y') \Longrightarrow C' \subseteq C$ **and**
D: $\bigwedge C\ C'\ Y\ Y'.\ \vdash c_2\ (\subseteq A,\ X) = (C,\ Y) \Longrightarrow$
$(U,\ \mathit{False}) \models c_2\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Y') \Longrightarrow C' \subseteq C$
shows $C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-if*:
assumes
A: $\vdash \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2\ (\subseteq A,\ X) = (C,\ Y)$ **and**
B: $(U,\ \mathit{False}) \models \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Y')$ **and**
C: $\bigwedge U'\ p\ B_1\ B_2\ C\ C'\ Y\ Y'.$
$(U',\ p) = (\mathit{insert}\ (\mathit{Univ?}\ A\ X,\ \mathit{bvars}\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = p \Longrightarrow \vdash c_1\ (\subseteq B_1,\ X) = (C,\ Y) \Longrightarrow$
$(U',\ \mathit{False}) \models c_1\ (\subseteq B_1,\ X) = \mathit{Some}\ (C',\ Y') \Longrightarrow C' \subseteq C$ **and**
D: $\bigwedge U'\ p\ B_1\ B_2\ C\ C'\ Y\ Y'.$
$(U',\ p) = (\mathit{insert}\ (\mathit{Univ?}\ A\ X,\ \mathit{bvars}\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = p \Longrightarrow \vdash c_2\ (\subseteq B_2,\ X) = (C,\ Y) \Longrightarrow$
$(U',\ \mathit{False}) \models c_2\ (\subseteq B_2,\ X) = \mathit{Some}\ (C',\ Y') \Longrightarrow C' \subseteq C$
shows $C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst-while*:
assumes
A: $\vdash \mathit{WHILE}\ b\ \mathit{DO}\ c\ (\subseteq A,\ X) = (B,\ Z)$ **and**
B: $(U,\ \mathit{False}) \models \mathit{WHILE}\ b\ \mathit{DO}\ c\ (\subseteq A,\ X) = \mathit{Some}\ (B',\ Z')$
shows $B' \subseteq B$
⟨*proof*⟩

lemma *ctyping1-ctyping2-fst*:
$[\![\vdash c\ (\subseteq A,\ X) = (C,\ Z);\ (U,\ \mathit{False}) \models c\ (\subseteq A,\ X) = \mathit{Some}\ (C',\ Z')]\!] \Longrightarrow$
$C' \subseteq C$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-skip* [*elim*!]:
$[\![\vdash SKIP\ (\subseteq A,\ X) = (C,\ Z);$
$(U,\ False) \models SKIP\ (\subseteq A,\ X) = Some\ (C',\ Z')]\!] \Longrightarrow$
$Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-assign* [*elim*!]:
$[\![\vdash x ::= a\ (\subseteq A,\ X) = (C,\ Z);$
$(U,\ False) \models x ::= a\ (\subseteq A,\ X) = Some\ (C',\ Z')]\!] \Longrightarrow$
$Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-input* [*elim*!]:
$[\![\vdash IN\ x\ (\subseteq A,\ X) = (C,\ Z);$
$(U,\ False) \models IN\ x\ (\subseteq A,\ X) = Some\ (C',\ Z')]\!] \Longrightarrow$
$Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-output* [*elim*!]:
$[\![\vdash OUT\ x\ (\subseteq A,\ X) = (C,\ Z);$
$(U,\ False) \models OUT\ x\ (\subseteq A,\ X) = Some\ (C',\ Z')]\!] \Longrightarrow$
$Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-seq*:
assumes
A: $\vdash c_1;;\ c_2\ (\subseteq A,\ X) = (C,\ Z)$ **and**
B: $(U,\ False) \models c_1;;\ c_2\ (\subseteq A,\ X) = Some\ (C',\ Z')$ **and**
C: $\bigwedge B\ B'\ Y\ Y'.\ \vdash c_1\ (\subseteq A,\ X) = (B,\ Y) \Longrightarrow$
$(U,\ False) \models c_1\ (\subseteq A,\ X) = Some\ (B',\ Y') \Longrightarrow Y \subseteq Y'$ **and**
D: $\bigwedge p\ B'\ Y'\ D'\ C'\ W'\ Z'.$
$(U,\ False) \models c_1\ (\subseteq A,\ X) = Some\ p \Longrightarrow (B',\ Y') = p \Longrightarrow$
$\vdash c_2\ (\subseteq B',\ Y') = (D',\ W') \Longrightarrow$
$(U,\ False) \models c_2\ (\subseteq B',\ Y') = Some\ (C',\ Z') \Longrightarrow W' \subseteq Z'$
shows $Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-or*:
assumes
A: $\vdash c_1\ OR\ c_2\ (\subseteq A,\ X) = (C,\ Y)$ **and**
B: $(U,\ False) \models c_1\ OR\ c_2\ (\subseteq A,\ X) = Some\ (C',\ Y')$ **and**
C: $\bigwedge C\ C'\ Y\ Y'.\ \vdash c_1\ (\subseteq A,\ X) = (C,\ Y) \Longrightarrow$
$(U,\ False) \models c_1\ (\subseteq A,\ X) = Some\ (C',\ Y') \Longrightarrow Y \subseteq Y'$ **and**
D: $\bigwedge C\ C'\ Y\ Y'.\ \vdash c_2\ (\subseteq A,\ X) = (C,\ Y) \Longrightarrow$
$(U,\ False) \models c_2\ (\subseteq A,\ X) = Some\ (C',\ Y') \Longrightarrow Y \subseteq Y'$
shows $Y \subseteq Y'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-if*:
assumes
A: $\vdash$ *IF* b *THEN* c_1 *ELSE* c_2 $(\subseteq A, X) = (C, Y)$ **and**
B: $(U, \mathit{False}) \models$ *IF* b *THEN* c_1 *ELSE* c_2 $(\subseteq A, X) = \mathit{Some}\ (C', Y')$ **and**
C: $\bigwedge U'\ p\ B_1\ B_2\ C\ C'\ Y\ Y'$.
$(U', p) = (\mathit{insert}\ (\mathit{Univ?}\ A\ X, \mathit{bvars}\ b)\ U, \models b\ (\subseteq A, X)) \Longrightarrow$
$(B_1, B_2) = p \Longrightarrow \vdash c_1\ (\subseteq B_1, X) = (C, Y) \Longrightarrow$
$(U', \mathit{False}) \models c_1\ (\subseteq B_1, X) = \mathit{Some}\ (C', Y') \Longrightarrow Y \subseteq Y'$ **and**
D: $\bigwedge U'\ p\ B_1\ B_2\ C\ C'\ Y\ Y'$.
$(U', p) = (\mathit{insert}\ (\mathit{Univ?}\ A\ X, \mathit{bvars}\ b)\ U, \models b\ (\subseteq A, X)) \Longrightarrow$
$(B_1, B_2) = p \Longrightarrow \vdash c_2\ (\subseteq B_2, X) = (C, Y) \Longrightarrow$
$(U', \mathit{False}) \models c_2\ (\subseteq B_2, X) = \mathit{Some}\ (C', Y') \Longrightarrow Y \subseteq Y'$
shows $Y \subseteq Y'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd-while*:
assumes
A: $\vdash$ *WHILE* b *DO* c $(\subseteq A, X) = (B, Z)$ **and**
B: $(U, \mathit{False}) \models$ *WHILE* b *DO* c $(\subseteq A, X) = \mathit{Some}\ (B', Z')$
shows $Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2-snd*:
$[\![\vdash c\ (\subseteq A, X) = (C, Z); (U, \mathit{False}) \models c\ (\subseteq A, X) = \mathit{Some}\ (C', Z')]\!] \Longrightarrow$
$Z \subseteq Z'$
⟨*proof*⟩

lemma *ctyping1-ctyping2*:
$[\![\vdash c\ (\subseteq A, X) = (C, Z); (U, \mathit{False}) \models c\ (\subseteq A, X) = \mathit{Some}\ (C', Z')]\!] \Longrightarrow$
$C' \subseteq C \wedge Z \subseteq Z'$
⟨*proof*⟩

lemma *btyping2-aux-approx-1* [*elim*]:
assumes
A: $\models b_1\ (\subseteq A, X) = \mathit{Some}\ B_1$ **and**
B: $\models b_2\ (\subseteq A, X) = \mathit{Some}\ B_2$ **and**
C: $\mathit{bval}\ b_1\ s$ **and**
D: $\mathit{bval}\ b_2\ s$ **and**
E: $r \in A$ **and**
F: $s = r\ (\subseteq \mathit{state} \cap X)$
shows $\exists r' \in B_1 \cap B_2.\ r = r'\ (\subseteq \mathit{state} \cap X)$
⟨*proof*⟩

lemma *btyping2-aux-approx-2* [*elim*]:
assumes
A: $\mathit{avars}\ a_1 \subseteq \mathit{state}$ **and**

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

lemma *btyping2-aux-approx-3* [*elim*]:
assumes
A: *avars* $a_1 \subseteq$ *state* **and**
B: *avars* $a_2 \subseteq$ *state* **and**
C: *avars* $a_1 \subseteq X$ **and**
D: *avars* $a_2 \subseteq X$ **and**
E: $\neg$ *aval* $a_1\ s <$ *aval* $a_2\ s$ **and**
F: $r \in A$ **and**
G: $s = r\ (\subseteq$ *state* $\cap\ X)$
shows $\exists\, r' \in A - \{s \in A.$ *aval* $a_1\ s <$ *aval* $a_2\ s\}.\ r = r'\ (\subseteq$ *state* $\cap\ X)$
⟨*proof*⟩

lemma *btyping2-aux-approx*:
$[\![\models b\ (\subseteq A,\ X) =$ *Some* $A';\ s \in$ *Univ* $A\ (\subseteq$ *state* $\cap\ X)]\!] \Longrightarrow$
$s \in$ *Univ* (*if bval* $b\ s$ *then* A' *else* $A - A')\ (\subseteq$ *state* $\cap\ X)$
⟨*proof*⟩

lemma *btyping2-approx*:
$[\![\models b\ (\subseteq A,\ X) = (B_1,\ B_2);\ s \in$ *Univ* $A\ (\subseteq$ *state* $\cap\ X)]\!] \Longrightarrow$
$s \in$ *Univ* (*if bval* $b\ s$ *then* B_1 *else* $B_2)\ (\subseteq$ *state* $\cap\ X)$
⟨*proof*⟩

lemma *ctyping2-approx-assign* [*elim*!]:
$[\![\forall\, t'.$ *aval* $a\ s = t'\ x \longrightarrow (\forall\, s.\ t' = s(x :=$ *aval* $a\ s) \longrightarrow s \notin A) \vee$
$(\exists\, y \in$ *state* $\cap\ X.\ y \neq x \wedge t\ y \neq t'\ y)$;
$v \models a\ (\subseteq X);\ t \in A;\ s = t\ (\subseteq$ *state* $\cap\ X)]\!] \Longrightarrow$ *False*
⟨*proof*⟩

lemma *ctyping2-approx-if-1*:
$[\![$*bval* $b\ s$; $\models b\ (\subseteq A,\ X) = (B_1,\ B_2);\ r \in A;\ s = r\ (\subseteq$ *state* $\cap\ X)$;
(*insert* (*Univ?* $A\ X$, *bvars* b) U, $v) \models c_1\ (\subseteq B_1,\ X) =$ *Some* $(C_1,\ Y_1)$;
$\bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_1\ (\subseteq A,\ X) =$ *Some* $(B,\ Y) \Longrightarrow$
$\exists\, r \in A.\ s = r\ (\subseteq$ *state* $\cap\ X) \Longrightarrow \exists\, r' \in B.\ t = r'\ (\subseteq$ *state* $\cap\ Y)]\!] \Longrightarrow$
$\exists\, r' \in C_1 \cup C_2.\ t = r'\ (\subseteq$ *state* $\cap\ (Y_1 \cap Y_2))$
⟨*proof*⟩

lemma *ctyping2-approx-if-2*:
$[\![\neg$ *bval* $b\ s$; $\models b\ (\subseteq A,\ X) = (B_1,\ B_2);\ r \in A;\ s = r\ (\subseteq$ *state* $\cap\ X)$;
(*insert* (*Univ?* $A\ X$, *bvars* b) U, $v) \models c_2\ (\subseteq B_2,\ X) =$ *Some* $(C_2,\ Y_2)$;

$\bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow$
$\exists\, r \in A.\ s = r\ (\subseteq \mathit{state} \cap X) \Longrightarrow \exists\, r' \in B.\ t = r'\ (\subseteq \mathit{state} \cap Y)]\!] \Longrightarrow$
$\exists\, r' \in C_1 \cup C_2.\ t = r'\ (\subseteq \mathit{state} \cap (Y_1 \cap Y_2))$
⟨*proof*⟩

lemma *ctyping2-approx-while-1* [*elim*]:
$[\![\neg\ \mathit{bval}\ b\ s;\ r \in A;\ s = r\ (\subseteq \mathit{state} \cap X);\ \models b\ (\subseteq A,\ X) = (B,\ \{\})]\!] \Longrightarrow$
$\exists\, t \in C.\ s = t\ (\subseteq \mathit{state} \cap Y)$
⟨*proof*⟩

lemma *ctyping2-approx-while-2* [*elim*]:
$[\![\forall\, t \in B_2 \cup B_2'.\ \exists\, x \in \mathit{state} \cap (X \cap Y).\ r\ x \neq t\ x;\ \neg\ \mathit{bval}\ b\ s;$
$r \in A;\ s = r\ (\subseteq \mathit{state} \cap X);\ \models b\ (\subseteq A,\ X) = (B_1,\ B_2)]\!] \Longrightarrow \mathit{False}$
⟨*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: $(\{\},\ \mathit{False}) \models c\ (\subseteq B_1,\ X) = \mathit{Some}\ (D,\ Z)$ **and**
E: $(\{\},\ \mathit{False}) \models c\ (\subseteq B_1',\ Y) = \mathit{Some}\ (D',\ Z')$ **and**
F: $r_1 \in A$ **and**
G: $s_1 = r_1\ (\subseteq \mathit{state} \cap X)$ **and**
H: $\mathit{bval}\ b\ s_1$ **and**
I: $\bigwedge C\ B\ Y\ W\ U.\ (\mathit{case} \models b\ (\subseteq C,\ Y)\ \mathit{of}\ (B_1',\ B_2') \Rightarrow$
$\mathit{case} \vdash c\ (\subseteq B_1',\ Y)\ \mathit{of}\ (C',\ Y') \Rightarrow$
$\mathit{case} \models b\ (\subseteq C',\ Y')\ \mathit{of}\ (B_1'',\ B_2'') \Rightarrow \mathit{if}$
$(\forall\, s \in \mathit{Univ?}\ C\ Y \cup \mathit{Univ?}\ C'\ Y'.\ \forall\, x \in \mathit{bvars}\ b.\ \forall\, y.\ s\colon \mathit{dom}\ x \rightsquigarrow \mathit{dom}\ y) \wedge$
$(\forall\, p \in U.\ \mathit{case}\ p\ \mathit{of}\ (B,\ W) \Rightarrow \forall\, s \in B.\ \forall\, x \in W.\ \forall\, y.\ s\colon \mathit{dom}\ x \rightsquigarrow \mathit{dom}\ y)$
$\mathit{then\ case}\ (\{\},\ \mathit{False}) \models c\ (\subseteq B_1',\ Y)\ \mathit{of}$
$\mathit{None} \Rightarrow \mathit{None} \mid \mathit{Some}\ \text{-} \Rightarrow \mathit{case}\ (\{\},\ \mathit{False}) \models c\ (\subseteq B_1'',\ Y')\ \mathit{of}$
$\mathit{None} \Rightarrow \mathit{None} \mid \mathit{Some}\ \text{-} \Rightarrow \mathit{Some}\ (B_2' \cup B_2'',\ \mathit{Univ??}\ B_2'\ Y \cap Y')$
$\mathit{else\ None}) = \mathit{Some}\ (B,\ W) \Longrightarrow$
$\exists\, r \in C.\ s_2 = r\ (\subseteq \mathit{state} \cap Y) \Longrightarrow \exists\, r \in B.\ s_3 = r\ (\subseteq \mathit{state} \cap W)$
(**is** $\bigwedge C\ B\ Y\ W\ U.\ ?P\ C\ B\ Y\ W\ U \Longrightarrow \text{-} \Longrightarrow \text{-}$) **and**
J: $\bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow$
$\exists\, r \in A.\ s_1 = r\ (\subseteq \mathit{state} \cap X) \Longrightarrow \exists\, r \in B.\ s_2 = r\ (\subseteq \mathit{state} \cap Y)$ **and**
K: $\forall\, s \in \mathit{Univ?}\ A\ X \cup \mathit{Univ?}\ C\ Y.\ \forall\, x \in \mathit{bvars}\ b.\ \forall\, y.\ s\colon \mathit{dom}\ x \rightsquigarrow \mathit{dom}\ y$ **and**
L: $\forall\, p \in U.\ \forall\, B\ W.\ p = (B,\ W) \longrightarrow$
$(\forall\, s \in B.\ \forall\, x \in W.\ \forall\, y.\ s\colon \mathit{dom}\ x \rightsquigarrow \mathit{dom}\ y)$
shows $\exists\, r \in B_2 \cup B_2'.\ s_3 = r\ (\subseteq \mathit{state} \cap \mathit{Univ??}\ B_2\ X \cap Y)$
⟨*proof*⟩

lemmas *ctyping2-approx-while-3* =
ctyping2-approx-while-aux [**where** $B_2 = \{\}$, *simplified*]

lemma *ctyping2-approx-while-4*:
$[\![\models b\ (\subseteq A,\ X) = (B_1,\ B_2);$

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

lemma *ctyping2-approx*:
$\llbracket(c,\ s,\ p) \Rightarrow (t,\ q);\ (U,\ v) \models c\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y);$
$\quad s \in \mathit{Univ}\ A\ (\subseteq \mathit{state} \cap X)\rrbracket \Longrightarrow t \in \mathit{Univ}\ B\ (\subseteq \mathit{state} \cap Y)$
⟨*proof*⟩

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*:

$s = t\ (\subseteq bvars\ b) \Longrightarrow bval\ b\ s = bval\ b\ t$
⟨*proof*⟩

lemma *eq-streams-subset*:
$[\![f = f'\ (\subseteq vs,\ vs',\ T);\ T' \subseteq T]\!] \Longrightarrow f = f'\ (\subseteq vs,\ vs',\ T')$
⟨*proof*⟩

lemma *flow-append-1*:
assumes *A*: $\bigwedge cfs' :: (com \times stage)\ list.$
$c\ \#\ map\ fst\ (cfs :: (com \times stage)\ list) = map\ fst\ cfs' \Longrightarrow$
$flow\text{-}aux\ (map\ fst\ cfs' @ map\ fst\ cfs'') =$
$flow\text{-}aux\ (map\ fst\ cfs') @ flow\text{-}aux\ (map\ fst\ cfs'')$
shows $flow\text{-}aux\ (c\ \#\ map\ fst\ cfs @ map\ fst\ cfs'') =$
$flow\text{-}aux\ (c\ \#\ map\ fst\ cfs) @ flow\text{-}aux\ (map\ fst\ cfs'')$
⟨*proof*⟩

lemma *flow-append*:
$flow\ (cfs @ cfs') = flow\ cfs @ flow\ cfs'$
⟨*proof*⟩

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

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

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

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

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

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

IN x ⇒ s(x := f x (length [p←vs. fst p = x])) |
- ⇒ s)
⟨*proof*⟩

lemma *run-flow-observe*:
run-flow (⟨X⟩ # cs) vs s f = run-flow cs vs s f
⟨*proof*⟩

lemma *out-flow-append*:
out-flow (cs @ cs′) vs s f =
out-flow cs vs s f @
out-flow cs′ (vs @ in-flow cs vs f) (run-flow cs vs s f) f
⟨*proof*⟩

lemma *out-flow-one*:
out-flow [c] vs s f = (case c of
OUT x ⇒ [(x, s x)] | - ⇒ [])
⟨*proof*⟩

lemma *no-upd-empty*:
no-upd cs {}
⟨*proof*⟩

lemma *no-upd-append*:
no-upd (cs @ cs′) X = (no-upd cs X ∧ no-upd cs′ X)
⟨*proof*⟩

lemma *no-upd-in-flow*:
no-upd cs X ⟹ [p←in-flow cs vs f. fst p ∈ X] = []
⟨*proof*⟩

lemma *no-upd-run-flow*:
no-upd cs X ⟹ run-flow cs vs s f = s (⊆ X)
⟨*proof*⟩

lemma *no-upd-out-flow*:
no-upd cs X ⟹ [p←out-flow cs vs s f. fst p ∈ X] = []
⟨*proof*⟩

lemma *small-stepsl-append*:
⟦cf →∗{cfs} cf′; cf′ →∗{cfs′} cf′′⟧ ⟹ cf →∗{cfs @ cfs′} cf′′
⟨*proof*⟩

lemma *small-step-stream*:
(c, s, f, vs, ws) → (c′, p) ⟹ ∃ s′ vs′ ws′. p = (s′, f, vs′, ws′)
⟨*proof*⟩

lemma *small-stepsl-stream*:
$(c,\ s,\ f,\ vs,\ ws) \rightarrow*\{cfs\}\ (c',\ p) \Longrightarrow \exists\, s'\ vs'\ ws'.\ p = (s',\ f,\ vs',\ ws')$
⟨*proof*⟩

lemma *small-steps-stepsl-1*:
$\exists\, cfs.\ cf \rightarrow*\{cfs\}\ cf$
⟨*proof*⟩

lemma *small-steps-stepsl-2*:
$[\![cf \rightarrow cf';\ cf' \rightarrow*\{cfs\}\ cf'']\!] \Longrightarrow \exists\, cfs'.\ cf \rightarrow*\{cfs'\}\ cf''$
⟨*proof*⟩

lemma *small-steps-stepsl*:
$cf \rightarrow* cf' \Longrightarrow \exists\, cfs.\ cf \rightarrow*\{cfs\}\ cf'$
⟨*proof*⟩

lemma *small-stepsl-steps*:
$cf \rightarrow*\{cfs\}\ cf' \Longrightarrow cf \rightarrow* cf'$
⟨*proof*⟩

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

lemma *small-stepsl-cons-1*:
$cf \rightarrow*\{[cf']\}\ cf'' \Longrightarrow cf' = cf \wedge (\exists\, cf'.\ cf \rightarrow cf' \wedge cf' \rightarrow*\{[]\}\ cf'')$
⟨*proof*⟩

lemma *small-stepsl-cons-2*:
$[\![cf \rightarrow*\{cf' \#\ cfs\}\ cf'' \Longrightarrow$
$\quad cf' = cf \wedge (\exists\, cf'.\ cf \rightarrow cf' \wedge cf' \rightarrow*\{cfs\}\ cf'');$
$\ cf \rightarrow*\{cf' \#\ cfs\ @\ [cf']\}\ cf'']\!] \Longrightarrow$
$\quad cf' = cf \wedge (\exists\, cf'.\ cf \rightarrow cf' \wedge cf' \rightarrow*\{cfs\ @\ [cf']\}\ cf'')$
⟨*proof*⟩

lemma *small-stepsl-cons*:
$cf \rightarrow*\{cf' \#\ cfs\}\ cf'' \Longrightarrow$
$\quad cf' = cf \wedge$
$\quad (\exists\, cf'.\ cf \rightarrow cf' \wedge cf' \rightarrow*\{cfs\}\ cf'')$
⟨*proof*⟩

lemma *small-stepsl-skip*:
$(SKIP,\ p) \rightarrow*\{cfs\}\ cf \Longrightarrow cf = (SKIP,\ p) \wedge flow\ cfs = []$
⟨*proof*⟩

lemma *small-stepsl-assign*:
$(x ::= a,\ s,\ p) \rightarrow*\{cfs\}\ cf \Longrightarrow$
$cf = (x ::= a,\ s,\ p) \wedge$
$flow\ cfs = [] \vee$
$cf = (SKIP,\ s(x := aval\ a\ s),\ p) \wedge$
$flow\ cfs = [x ::= a]$
⟨*proof*⟩

lemma *small-stepsl-input*:
$(IN\ x,\ s,\ f,\ vs,\ ws) \rightarrow*\{cfs\}\ cf \Longrightarrow$
$cf = (IN\ x,\ s,\ f,\ vs,\ ws) \wedge$
$flow\ cfs = [] \vee$
$(let\ n = length\ [p \leftarrow vs.\ fst\ p = x]$
$in\ cf = (SKIP,\ s(x := f\ x\ n),\ f,\ vs\ @\ [(x,\ f\ x\ n)],\ ws) \wedge$
$flow\ cfs = [IN\ x])$
⟨*proof*⟩

lemma *small-stepsl-output*:
$(OUT\ x,\ s,\ f,\ vs,\ ws) \rightarrow*\{cfs\}\ cf \Longrightarrow$
$cf = (OUT\ x,\ s,\ f,\ vs,\ ws) \wedge$
$flow\ cfs = [] \vee$
$cf = (SKIP,\ s,\ f,\ vs,\ ws\ @\ [(x,\ s\ x)]) \wedge$
$flow\ cfs = [OUT\ x]$
⟨*proof*⟩

lemma *small-stepsl-seq-1*:
$(c_1;;\ c_2,\ p) \rightarrow*\{[]\}\ (c,\ q) \Longrightarrow$
$(\exists\, c'\ cfs'.\ c = c';;\ c_2 \wedge$
$(c_1,\ p) \rightarrow*\{cfs'\}\ (c',\ q) \wedge$
$flow\ [] = flow\ cfs') \vee$
$(\exists\, p'\ cfs'\ cfs''.\ length\ cfs'' < length\ [] \wedge$
$(c_1,\ p) \rightarrow*\{cfs'\}\ (SKIP,\ p') \wedge$
$(c_2,\ p') \rightarrow*\{cfs''\}\ (c,\ q) \wedge$
$flow\ [] = flow\ cfs' @ flow\ cfs'')$
⟨*proof*⟩

lemma *small-stepsl-seq-2*:
assumes *A*: $\bigwedge c'\ q'.\ cf = (c',\ q') \Longrightarrow$
$(c_1;;\ c_2,\ p) \rightarrow*\{cfs\}\ (c',\ q') \Longrightarrow$
$(\exists\, c''\ cfs'.\ c' = c'';;\ c_2 \wedge$
$(c_1,\ p) \rightarrow*\{cfs'\}\ (c'',\ q') \wedge$
$flow\ cfs = flow\ cfs') \vee$
$(\exists\, p'\ cfs'\ cfs''.\ length\ cfs'' < length\ cfs \wedge$
$(c_1,\ p) \rightarrow*\{cfs'\}\ (SKIP,\ p') \wedge$
$(c_2,\ p') \rightarrow*\{cfs''\}\ (c',\ q') \wedge$
$flow\ cfs = flow\ cfs' @ flow\ cfs'')$
(**is** $\bigwedge c'\ q'.$ - $\Longrightarrow$ - $\Longrightarrow$
$(\exists\, c''\ cfs'.\ ?P\ c'\ q'\ c''\ cfs') \vee$

$(\exists\, p'\ cfs'\ cfs''.\ ?Q\ c'\ q'\ p'\ cfs'\ cfs''))$
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** $?T \vee ?U$)
⟨*proof*⟩

lemma *small-stepsl-seq*:
$(c_1;;\ c_2,\ p) \rightarrow* \{cfs\}\ (c,\ q) \Longrightarrow$
$(\exists\, c'\ cfs'.\ c = c';;\ c_2 \wedge$
$(c_1,\ p) \rightarrow* \{cfs'\}\ (c',\ q) \wedge$
$flow\ cfs = flow\ cfs') \vee$
$(\exists\, p'\ cfs'\ cfs''.\ length\ cfs'' < length\ cfs \wedge$
$(c_1,\ p) \rightarrow* \{cfs'\}\ (SKIP,\ p') \wedge (c_2,\ p') \rightarrow* \{cfs''\}\ (c,\ q) \wedge$
$flow\ cfs = flow\ cfs' @ flow\ cfs'')$
⟨*proof*⟩

lemma *small-stepsl-or-1*:
assumes A: $(c_1\ OR\ c_2,\ p) \rightarrow* \{cfs\}\ cf \Longrightarrow$
$cf = (c_1\ OR\ c_2,\ p) \wedge$
$flow\ cfs = [] \vee$
$(c_1,\ p) \rightarrow* \{tl\ cfs\}\ cf \wedge$
$flow\ cfs = flow\ (tl\ cfs) \vee$
$(c_2,\ p) \rightarrow* \{tl\ cfs\}\ cf \wedge$
$flow\ cfs = flow\ (tl\ cfs)$
(**is** - $\Longrightarrow ?P \vee ?Q \vee ?R$)
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** - $\vee ?T$)
⟨*proof*⟩

lemma *small-stepsl-or*:
$(c_1\ OR\ c_2,\ p) \rightarrow* \{cfs\}\ cf \Longrightarrow$
$cf = (c_1\ OR\ c_2,\ p) \wedge$
$flow\ cfs = [] \vee$
$(c_1,\ p) \rightarrow* \{tl\ cfs\}\ cf \wedge$

flow cfs = *flow* (*tl cfs*) ∨
(*c*$_2$, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = *flow* (*tl cfs*)
⟨*proof*⟩

lemma *small-stepsl-if-1*:
assumes *A*: (*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) →∗{*cfs*} *cf* ⟹
cf = (*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) ∧
flow cfs = [] ∨
bval b s ∧ (*c*$_1$, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*) ∨
¬ *bval b s* ∧ (*c*$_2$, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*)
(**is** - ⟹ *?P* ∨ *?Q* ∨ *?R*)
assumes *B*: (*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) →∗{*cfs* @ [*cf*]} *cf′*
shows
cf′ = (*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) ∧
flow (*cfs* @ [*cf*]) = [] ∨
bval b s ∧ (*c*$_1$, *s*, *p*) →∗{*tl* (*cfs* @ [*cf*])} *cf′* ∧
flow (*cfs* @ [*cf*]) = ⟨*bvars b*⟩ # *flow* (*tl* (*cfs* @ [*cf*])) ∨
¬ *bval b s* ∧ (*c*$_2$, *s*, *p*) →∗{*tl* (*cfs* @ [*cf*])} *cf′* ∧
flow (*cfs* @ [*cf*]) = ⟨*bvars b*⟩ # *flow* (*tl* (*cfs* @ [*cf*]))
(**is** - ∨ *?T*)
⟨*proof*⟩

lemma *small-stepsl-if*:
(*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) →∗{*cfs*} *cf* ⟹
cf = (*IF b THEN c*$_1$ *ELSE c*$_2$, *s*, *p*) ∧
flow cfs = [] ∨
bval b s ∧ (*c*$_1$, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*) ∨
¬ *bval b s* ∧ (*c*$_2$, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*)
⟨*proof*⟩

lemma *small-stepsl-while-1*:
assumes *A*: (*WHILE b DO c*, *s*, *p*) →∗{*cfs*} *cf* ⟹
cf = (*WHILE b DO c*, *s*, *p*) ∧
flow cfs = [] ∨
bval b s ∧ (*c*;; *WHILE b DO c*, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*) ∨
¬ *bval b s* ∧ *cf* = (*SKIP*, *s*, *p*) ∧
flow cfs = [⟨*bvars b*⟩]
(**is** - ⟹ *?P* ∨ *?Q* ∨ *?R*)
assumes *B*: (*WHILE b DO c*, *s*, *p*) →∗{*cfs* @ [*cf*]} *cf′*
shows
cf′ = (*WHILE b DO c*, *s*, *p*) ∧

flow (*cfs* @ [*cf*]) = [] ∨
bval b s ∧ (*c*;; *WHILE b DO c*, *s*, *p*) →∗{*tl* (*cfs* @ [*cf*])} *cf′* ∧
flow (*cfs* @ [*cf*]) = ⟨*bvars b*⟩ # *flow* (*tl* (*cfs* @ [*cf*])) ∨
¬ *bval b s* ∧ *cf′* = (*SKIP*, *s*, *p*) ∧
flow (*cfs* @ [*cf*]) = [⟨*bvars b*⟩]
(**is** - ∨ *?T*)
⟨*proof*⟩

lemma *small-stepsl-while*:
(*WHILE b DO c*, *s*, *p*) →∗{*cfs*} *cf* ⟹
cf = (*WHILE b DO c*, *s*, *p*) ∧
flow cfs = [] ∨
bval b s ∧ (*c*;; *WHILE b DO c*, *s*, *p*) →∗{*tl cfs*} *cf* ∧
flow cfs = ⟨*bvars b*⟩ # *flow* (*tl cfs*) ∨
¬ *bval b s* ∧ *cf* = (*SKIP*, *s*, *p*) ∧
flow cfs = [⟨*bvars b*⟩]
⟨*proof*⟩

lemma *small-steps-in-flow-1*:
⟦(*c*, *s*, *f*, *vs*, *ws*) → (*c′*, *s′*, *f′*, *vs′*, *ws′*);
vs″ = *vs′* @ *drop* (*length vs′*) *vs″*⟧ ⟹
vs″ = *vs* @ *drop* (*length vs*) *vs″*
⟨*proof*⟩

lemma *small-steps-in-flow*:
(*c*, *s*, *f*, *vs*, *ws*) →∗ (*c′*, *s′*, *f′*, *vs′*, *ws′*) ⟹
vs′ = *vs* @ *drop* (*length vs*) *vs′*
⟨*proof*⟩

lemma *small-steps-out-flow-1*:
⟦(*c*, *s*, *f*, *vs*, *ws*) → (*c′*, *s′*, *f′*, *vs′*, *ws′*);
ws″ = *ws′* @ *drop* (*length ws′*) *ws″*⟧ ⟹
ws″ = *ws* @ *drop* (*length ws*) *ws″*
⟨*proof*⟩

lemma *small-steps-out-flow*:
(*c*, *s*, *f*, *vs*, *ws*) →∗ (*c′*, *s′*, *f′*, *vs′*, *ws′*) ⟹
ws′ = *ws* @ *drop* (*length ws*) *ws′*
⟨*proof*⟩

lemma *small-stepsl-in-flow-1*:
assumes
A: (*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*} (*c′*, *s′*, *f′*, *vs* @ *vs′*, *ws′*) **and**
B: (*c′*, *s′*, *f′*, *vs* @ *vs′*, *ws′*) → (*c″*, *s″*, *f″*, *vs″*, *ws″*)
shows *vs″* = *vs* @ *vs′* @
in-flow (*flow* [(*c′*, *s′*, *f′*, *vs* @ *vs′*, *ws′*)]) (*vs* @ *vs′*) *f*

⟨*proof*⟩

lemma *small-stepsl-in-flow*:
(*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*} (*c′*, *s′*, *f′*, *vs′*, *ws′*) ⟹
vs′ = *vs* @ *in-flow* (*flow cfs*) *vs f*
⟨*proof*⟩

lemma *small-stepsl-run-flow-1*:
assumes
A: (*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*}
(*c′*, *run-flow* (*flow cfs*) *vs s f*, *f′*, *vs′*, *ws′*) **and**
B: (*c′*, *run-flow* (*flow cfs*) *vs s f*, *f′*, *vs′*, *ws′*) →
(*c″*, *s″*, *f″*, *vs″*, *ws″*)
shows *s″* = *run-flow* (*flow* [(*c′*, *run-flow* (*flow cfs*) *vs s f*, *f′*, *vs′*, *ws′*)])
(*vs* @ *in-flow* (*flow cfs*) *vs f*) (*run-flow* (*flow cfs*) *vs s f*) *f*
⟨*proof*⟩

lemma *small-stepsl-run-flow*:
(*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*} (*c′*, *s′*, *f′*, *vs′*, *ws′*) ⟹
s′ = *run-flow* (*flow cfs*) *vs s f*
⟨*proof*⟩

lemma *small-stepsl-out-flow-1*:
assumes
A: (*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*} (*c′*, *s′*, *f′*, *vs′*, *ws* @ *ws′*) **and**
B: (*c′*, *s′*, *f′*, *vs′*, *ws* @ *ws′*) → (*c″*, *s″*, *f″*, *vs″*, *ws″*)
shows *ws″* = *ws* @ *ws′* @
out-flow (*flow* [(*c′*, *s′*, *f′*, *vs′*, *ws* @ *ws′*)]) (*vs* @ *in-flow* (*flow cfs*) *vs f*)
(*run-flow* (*flow cfs*) *vs s f*) *f*
⟨*proof*⟩

lemma *small-stepsl-out-flow*:
(*c*, *s*, *f*, *vs*, *ws*) →∗{*cfs*} (*c′*, *s′*, *f′*, *vs′*, *ws′*) ⟹
ws′ = *ws* @ *out-flow* (*flow cfs*) *vs s f*
⟨*proof*⟩

lemma *small-steps-inputs*:
assumes
A: (*c*, *s*, *f*, *vs*$_0$, *ws*$_0$) →∗{*cfs*$_1$} (*c*$_0$, *s*$_1$, *f*, *vs*$_1$, *ws*$_1$) **and**
B: (*c*$_1$, *s*$_1$, *f*, *vs*$_1$, *ws*$_1$) →∗{*cfs*$_2$} (*c*$_2$, *s*$_2$, *f*, *vs*$_2$, *ws*$_2$) **and**
C: (*c*, *s′*, *f′*, *vs*$_0$′, *ws*$_0$′) →∗ (*c*$_0$′, *s*$_1$′, *f′*, *vs*$_1$′, *ws*$_1$′) **and**
D: (*c*$_1$′, *s*$_1$′, *f′*, *vs*$_1$′, *ws*$_1$′) →∗ (*c*$_2$′, *s*$_2$′, *f′*, *vs*$_2$′, *ws*$_2$′) **and**
E: *map fst* [*p*←*drop* (*length vs*$_0$) *vs*$_1$. *P p*] =
map fst [*p*←*drop* (*length vs*$_0$′) *vs*$_1$′. *P p*] **and**
F: *map fst* [*p*←*drop* (*length vs*$_1$) *vs*$_2$. *P p*] =
map fst [*p*←*drop* (*length vs*$_1$′) *vs*$_2$′. *P p*]

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

lemma *small-steps-outputs*:
assumes
A: (c, s, f, vs_0, ws_0) →∗{cfs_1} (c_0, s_1, f, vs_1, ws_1) **and**
B: (c_1, s_1, f, vs_1, ws_1) →∗{cfs_2} (c_2, s_2, f, vs_2, ws_2) **and**
C: (c, s', f', $vs_0{}'$, $ws_0{}'$) →∗ ($c_0{}'$, $s_1{}'$, f', $vs_1{}'$, $ws_1{}'$) **and**
D: ($c_1{}'$, $s_1{}'$, f', $vs_1{}'$, $ws_1{}'$) →∗ ($c_2{}'$, $s_2{}'$, f', $vs_2{}'$, $ws_2{}'$) **and**
E: [p←*drop* (*length* ws_0) ws_1. P p] =
[p←*drop* (*length* $ws_0{}'$) $ws_1{}'$. P p] **and**
F: [p←*drop* (*length* ws_1) ws_2. P p] =
[p←*drop* (*length* $ws_1{}'$) $ws_2{}'$. P p]
shows [p←*drop* (*length* ws_0) ws_2. P p] =
[p←*drop* (*length* $ws_0{}'$) $ws_2{}'$. P p]
⟨*proof*⟩

5.2 Local context proofs

context *noninterf*
begin

lemma *no-upd-sources*:
no-upd cs X ⟹ ∀ x ∈ X. x ∈ *sources cs vs s f x*
⟨*proof*⟩

lemma *sources-aux-append*:
sources-aux cs vs s f x ⊆ *sources-aux* (*cs* @ *cs'*) *vs s f x*
⟨*proof*⟩

lemma *sources-out-append*:
sources-out cs vs s f x ⊆ *sources-out* (*cs* @ *cs'*) *vs s f x*
⟨*proof*⟩

lemma *sources-aux-sources*:
sources-aux cs vs s f x ⊆ *sources cs vs s f x*
⟨*proof*⟩

lemma *sources-aux-sources-out*:
sources-aux cs vs s f x ⊆ *sources-out cs vs s f x*
⟨*proof*⟩

lemma *sources-aux-observe-hd-1*:
∀ y ∈ X. s: *dom* y ⇝ *dom* x ⟹ X ⊆ *sources-aux* [⟨X⟩] *vs s f x*
⟨*proof*⟩

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

$[\![\forall\, y \in X.\ s\colon dom\ y \rightsquigarrow dom\ x \Longrightarrow X \subseteq sources\text{-}aux\ (\langle X\rangle\ \#\ xs)\ vs\ s\ f\ x;$
$\quad \forall\, y \in X.\ s\colon dom\ y \rightsquigarrow dom\ x]\!] \Longrightarrow$
$\ X \subseteq sources\text{-}aux\ (\langle X\rangle\ \#\ xs\ @\ [x'])\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *sources-aux-observe-hd*:
$\forall\, y \in X.\ s\colon dom\ y \rightsquigarrow dom\ x \Longrightarrow X \subseteq sources\text{-}aux\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *sources-aux-bval*:
assumes
$A\colon S \subseteq \{x.\ s = t\ (\subseteq sources\text{-}aux\ (\langle bvars\ b\rangle\ \#\ cs)\ vs\ s\ f\ x)\}$ **and**
$B\colon s \in Univ\ A\ (\subseteq state \cap X)$ **and**
$C\colon bval\ b\ s \neq bval\ b\ t$
shows $Univ?\ A\ X\colon bvars\ b \rightsquigarrow\mid S$
⟨*proof*⟩

lemma *ok-flow-aux-degen*:
assumes $A\colon \nexists S.\ S \neq \{\} \wedge S \subseteq \{x.\ s_1 = t_1\ (\subseteq sources\text{-}aux\ cs\ vs_1\ s_1\ f\ x)\}$
shows $\forall\, c_2'\ t_2\ vs_2'\ ws_2'.$
$ok\text{-}flow\text{-}aux\text{-}1\ c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ cs \wedge$
$ok\text{-}flow\text{-}aux\text{-}2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ cs \wedge$
$ok\text{-}flow\text{-}aux\text{-}3\ s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ cs$
(**is** $\forall\, c_2'\ t_2\ vs_2'\ ws_2'.\ ?P1\ c_2'\ t_2\ vs_2'\ ws_2' \wedge ?P2\ t_2 \wedge ?P3\ ws_2'$)
⟨*proof*⟩

lemma *tags-aux-append*:
$tags\text{-}aux\ cs\ vs\ s\ f\ x \subseteq tags\text{-}aux\ (cs\ @\ cs')\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-out-append*:
$tags\text{-}out\ cs\ vs\ s\ f\ x \subseteq tags\text{-}out\ (cs\ @\ cs')\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-aux-tags*:
$tags\text{-}aux\ cs\ vs\ s\ f\ x \subseteq tags\ cs\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-aux-tags-out*:
$tags\text{-}aux\ cs\ vs\ s\ f\ x \subseteq tags\text{-}out\ cs\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-ubound-1*:
assumes
$A\colon (y,\ Suc\ (length\ [c{\leftarrow}cs.\ c = IN\ y] + n)) \in tags\text{-}aux\ cs\ vs\ s\ f\ x$ **and**
$B\colon \bigwedge z\ n.\ y = z \Longrightarrow$
$\quad (z,\ length\ [c{\leftarrow}cs.\ c = IN\ z] + n) \notin tags\text{-}aux\ cs\ vs\ s\ f\ x$

shows *False*
⟨*proof*⟩

lemma *tags-ubound-2*:
assumes
A: $(y, Suc\ (length\ [c{\leftarrow}cs.\ c = IN\ y] + n)) \in tags\ cs\ vs\ s\ f\ x$ **and**
B: $\bigwedge z\ n.\ y = z \Longrightarrow z \neq x \Longrightarrow$
$(z, length\ [c{\leftarrow}cs.\ c = IN\ z] + n) \notin tags\ cs\ vs\ s\ f\ x$ **and**
C: $y \neq x$
shows *False*
⟨*proof*⟩

lemma *tags-ubound*:
$(y, length\ [c{\leftarrow}cs.\ c = IN\ y] + n) \notin tags\ cs\ vs\ s\ f\ x$
and *tags-aux-ubound*:
$(y, length\ [c{\leftarrow}cs.\ c = IN\ y] + n) \notin tags\text{-}aux\ cs\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-out-ubound-1*:
assumes
A: $(y, Suc\ (length\ [c{\leftarrow}cs.\ c = IN\ y] + n)) \in tags\text{-}out\ cs\ vs\ s\ f\ x$ **and**
B: $\bigwedge z\ n.\ y = z \Longrightarrow$
$(z, length\ [c{\leftarrow}cs.\ c = IN\ z] + n) \notin tags\text{-}out\ cs\ vs\ s\ f\ x$
shows *False*
⟨*proof*⟩

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

lemma *tags-less*:
$(y, n) \in tags\ cs\ vs\ s\ f\ x \Longrightarrow n < length\ [c{\leftarrow}cs.\ c = IN\ y]$
⟨*proof*⟩

lemma *tags-aux-less*:
$(y, n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow n < length\ [c{\leftarrow}cs.\ c = IN\ y]$
⟨*proof*⟩

lemma *tags-out-less*:
$(y, n) \in tags\text{-}out\ cs\ vs\ s\ f\ x \Longrightarrow n < length\ [c{\leftarrow}cs.\ c = IN\ y]$
⟨*proof*⟩

lemma *sources-observe-tl-1*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$sources\text{-}aux\ cs\ vs\ s\ f\ x \subseteq sources\text{-}aux\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x$ **and**

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

lemma *sources-observe-tl-2*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: \textit{com-flow}) \Longrightarrow$
sources-aux cs vs s f x $\subseteq$ *sources-aux* $(\langle X\rangle \# cs)$ *vs s f x* **and**
B: $\bigwedge z.\ c = (IN\ z :: \textit{com-flow}) \Longrightarrow$
sources-aux cs vs s f x $\subseteq$ *sources-aux* $(\langle X\rangle \# cs)$ *vs s f x* **and**
C: $\bigwedge z.\ c = (OUT\ z :: \textit{com-flow}) \Longrightarrow$
sources-aux cs vs s f x $\subseteq$ *sources-aux* $(\langle X\rangle \# cs)$ *vs s f x* **and**
D: $\bigwedge Y.\ c = \langle Y\rangle \Longrightarrow$
sources-aux cs vs s f x $\subseteq$ *sources-aux* $(\langle X\rangle \# cs)$ *vs s f x* **and**
E: $\bigwedge Y\ b\ w.\ c = \langle Y\rangle \Longrightarrow$
sources cs vs s f w $\subseteq$ *sources* $(\langle X\rangle \# cs)$ *vs s f w*
shows *sources-aux* $(cs\ @\ [c])$ *vs s f x* $\subseteq$
sources-aux $(\langle X\rangle \# cs\ @\ [c])$ *vs s f x*
(**is** - $\subseteq$ *?F c*)
⟨*proof*⟩

lemma *sources-observe-tl*:
sources cs vs s f x $\subseteq$ *sources* $(\langle X\rangle \# cs)$ *vs s f x*
and *sources-aux-observe-tl*:
sources-aux cs vs s f x $\subseteq$ *sources-aux* $(\langle X\rangle \# cs)$ *vs s f x*
⟨*proof*⟩

lemma *sources-out-observe-tl-1*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: \textit{com-flow}) \Longrightarrow$
sources-out cs vs s f x $\subseteq$ *sources-out* $(\langle X\rangle \# cs)$ *vs s f x* **and**
B: $\bigwedge z.\ c = (IN\ z :: \textit{com-flow}) \Longrightarrow$
sources-out cs vs s f x $\subseteq$ *sources-out* $(\langle X\rangle \# cs)$ *vs s f x* **and**
C: $\bigwedge z.\ c = (OUT\ z :: \textit{com-flow}) \Longrightarrow$
sources-out cs vs s f x $\subseteq$ *sources-out* $(\langle X\rangle \# cs)$ *vs s f x* **and**
D: $\bigwedge Y.\ c = \langle Y\rangle \Longrightarrow$

sources-out cs vs s f x ⊆ *sources-out* (⟨*X*⟩ # *cs*) *vs s f x*
shows *sources-out* (*cs* @ [*c*]) *vs s f x* ⊆
sources-out (⟨*X*⟩ # *cs* @ [*c*]) *vs s f x*
(**is** - ⊆ *?F c*)
⟨*proof*⟩

lemma *sources-out-observe-tl*:
sources-out cs vs s f x ⊆ *sources-out* (⟨*X*⟩ # *cs*) *vs s f x*
⟨*proof*⟩

lemma *tags-observe-tl-1*:
⟦⋀*z a*. *c* = *z* ::= *a* ⟹ *z* = *x* ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*z a b w*. *c* = *z* ::= *a* ⟹ *z* = *x* ⟹
tags (⟨*X*⟩ # *cs*) *vs s f w* = *tags cs vs s f w*;
⋀*z a*. *c* = *z* ::= *a* ⟹ *z* ≠ *x* ⟹
tags (⟨*X*⟩ # *cs*) *vs s f x* = *tags cs vs s f x*;
⋀*z*. *c* = *IN z* ⟹ *z* = *x* ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*z*. *c* = *IN z* ⟹ *z* ≠ *x* ⟹
tags (⟨*X*⟩ # *cs*) *vs s f x* = *tags cs vs s f x*;
⋀*z*. *c* = *OUT z* ⟹
tags (⟨*X*⟩ # *cs*) *vs s f x* = *tags cs vs s f x*;
⋀*Y b w*. *c* = ⟨*Y*⟩ ⟹
tags (⟨*X*⟩ # *cs*) *vs s f w* = *tags cs vs s f w*⟧ ⟹
tags (⟨*X*⟩ # *cs* @ [*c*]) *vs s f x* = *tags* (*cs* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *tags-observe-tl-2*:
⟦⋀*z a*. *c* = *z* ::= *a* ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*z*. *c* = *IN z* ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*z*. *c* = *OUT z* ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*Y*. *c* = ⟨*Y*⟩ ⟹
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*;
⋀*Y b w*. *c* = ⟨*Y*⟩ ⟹
tags (⟨*X*⟩ # *cs*) *vs s f w* = *tags cs vs s f w*⟧ ⟹
tags-aux (⟨*X*⟩ # *cs* @ [*c*]) *vs s f x* = *tags-aux* (*cs* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *tags-observe-tl*:
tags (⟨*X*⟩ # *cs*) *vs s f x* = *tags cs vs s f x*
and *tags-aux-observe-tl*:
tags-aux (⟨*X*⟩ # *cs*) *vs s f x* = *tags-aux cs vs s f x*
⟨*proof*⟩

lemma *tags-out-observe-tl-1*:
$[\![\bigwedge z\ a.\ c = z ::= a \Longrightarrow$
$\quad tags\text{-}out\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x = tags\text{-}out\ cs\ vs\ s\ f\ x;$
$\bigwedge z.\ c = IN\ z \Longrightarrow$
$\quad tags\text{-}out\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x = tags\text{-}out\ cs\ vs\ s\ f\ x;$
$\bigwedge z.\ c = OUT\ z \Longrightarrow$
$\quad tags\text{-}out\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x = tags\text{-}out\ cs\ vs\ s\ f\ x;$
$\bigwedge Y.\ c = \langle Y\rangle \Longrightarrow$
$\quad tags\text{-}out\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x = tags\text{-}out\ cs\ vs\ s\ f\ x]\!] \Longrightarrow$
$tags\text{-}out\ (\langle X\rangle\ \#\ cs\ @\ [c])\ vs\ s\ f\ x = tags\text{-}out\ (cs\ @\ [c])\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-out-observe-tl*:
$tags\text{-}out\ (\langle X\rangle\ \#\ cs)\ vs\ s\ f\ x = tags\text{-}out\ cs\ vs\ s\ f\ x$
⟨*proof*⟩

lemma *tags-sources-1*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$(y,\ n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow$
$let\ m = Suc\ (Max\ \{k.\ k \le length\ cs \wedge$
$length\ [c{\leftarrow}take\ k\ cs.\ c = IN\ y] \le n\})$
$in\ y \in sources\text{-}aux\ (drop\ m\ cs)\ (vs\ @\ in\text{-}flow\ (take\ m\ cs)\ vs\ f)$
$(run\text{-}flow\ (take\ m\ cs)\ vs\ s\ f)\ f\ x$
(**is** $\bigwedge$- -. - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ $let\ m = Suc\ (Max\ (?F\ cs))\ in$
- $\in sources\text{-}aux$ - $(?G\ m\ cs)\ (?H\ m\ cs)$ - -)
assumes
B: $\bigwedge z\ a\ b\ w.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$(y,\ n) \in tags\ cs\ vs\ s\ f\ w \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ w$ **and**
C: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z \ne x \Longrightarrow$
$(y,\ n) \in tags\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
D: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$(y,\ n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\text{-}aux\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
E: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow z \ne x \Longrightarrow$
$(y,\ n) \in tags\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
F: $\bigwedge z.\ c = (OUT\ z :: com\text{-}flow) \Longrightarrow$
$(y,\ n) \in tags\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
G: $\bigwedge X\ b\ w.\ c = \langle X\rangle \Longrightarrow$
$(y,\ n) \in tags\ cs\ vs\ s\ f\ w \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ w$ **and**
H: $(y,\ n) \in tags\ (cs\ @\ [c])\ vs\ s\ f\ x$
shows $let\ m = Suc\ (Max\ (?F\ (cs\ @\ [c])))\ in$
$y \in sources\ (drop\ m\ (cs\ @\ [c]))\ (?G\ m\ (cs\ @\ [c]))\ (?H\ m\ (cs\ @\ [c]))\ f\ x$

⟨*proof*⟩

lemma *tags-sources-2*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow$
$(y, n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow$
$let\ m = Suc\ (Max\ \{k.\ k \leq length\ cs \wedge$
$length\ [c \leftarrow take\ k\ cs.\ c = IN\ y] \leq n\})$
$in\ y \in sources\text{-}aux\ (drop\ m\ cs)\ (vs\ @\ in\text{-}flow\ (take\ m\ cs)\ vs\ f)$
$(run\text{-}flow\ (take\ m\ cs)\ vs\ s\ f)\ f\ x$
(**is** $\bigwedge$- -. - $\Longrightarrow$ - $\Longrightarrow$ $let\ m = Suc\ (Max\ (?F\ cs))\ in$
- $\in sources\text{-}aux$ - $(?G\ m\ cs)\ (?H\ m\ cs)$ - -)
assumes
B: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow$
$(y, n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\text{-}aux\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
C: $\bigwedge z.\ c = (OUT\ z :: com\text{-}flow) \Longrightarrow$
$(y, n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\text{-}aux\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
D: $\bigwedge X.\ c = \langle X \rangle \Longrightarrow$
$(y, n) \in tags\text{-}aux\ cs\ vs\ s\ f\ x \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\text{-}aux\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ x$ **and**
E: $\bigwedge X\ b\ w.\ c = \langle X \rangle \Longrightarrow$
$(y, n) \in tags\ cs\ vs\ s\ f\ w \Longrightarrow let\ m = Suc\ (Max\ (?F\ cs))\ in$
$y \in sources\ (drop\ m\ cs)\ (?G\ m\ cs)\ (?H\ m\ cs)\ f\ w$ **and**
F: $(y, n) \in tags\text{-}aux\ (cs\ @\ [c])\ vs\ s\ f\ x$
shows $let\ m = Suc\ (Max\ (?F\ (cs\ @\ [c])))\ in$
$y \in sources\text{-}aux\ (drop\ m\ (cs\ @\ [c]))\ (?G\ m\ (cs\ @\ [c]))\ (?H\ m\ (cs\ @\ [c]))\ f\ x$
⟨*proof*⟩

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

lemma *tags-out-sources-out-1*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow$
$(y, n) \in tags\text{-}out\ cs\ vs\ s\ f\ x \Longrightarrow$

let m = Suc (Max {k. k ≤ length cs ∧
length [c←take k cs. c = IN y] ≤ n})
in y ∈ sources-out (drop m cs) (vs @ in-flow (take m cs) vs f)
(run-flow (take m cs) vs s f) f x
(is ⋀- -. - ⟹ - ⟹ *let m = Suc (Max (?F cs)) in*
- ∈ sources-out - (?G m cs) (?H m cs) - -)

assumes
B: ⋀*z. c = (IN z :: com-flow)* ⟹
(y, n) ∈ tags-out cs vs s f x ⟹ *let m = Suc (Max (?F cs)) in*
y ∈ sources-out (drop m cs) (?G m cs) (?H m cs) f x **and**
C: ⋀*z. c = (OUT z :: com-flow)* ⟹
(y, n) ∈ tags-out cs vs s f x ⟹ *let m = Suc (Max (?F cs)) in*
y ∈ sources-out (drop m cs) (?G m cs) (?H m cs) f x **and**
D: ⋀*X. c = ⟨X⟩* ⟹
(y, n) ∈ tags-out cs vs s f x ⟹ *let m = Suc (Max (?F cs)) in*
y ∈ sources-out (drop m cs) (?G m cs) (?H m cs) f x **and**
E: *(y, n) ∈ tags-out (cs @ [c]) vs s f x*

shows *let m = Suc (Max (?F (cs @ [c]))) in*
y ∈ sources-out (drop m (cs @ [c])) (?G m (cs @ [c])) (?H m (cs @ [c])) f x
⟨*proof*⟩

lemma *tags-out-sources-out*:
(y, n) ∈ tags-out cs vs s f x ⟹
let m = Suc (Max {k. k ≤ length cs ∧
length [c←take k cs. c = IN y] ≤ n})
in y ∈ sources-out (drop m cs) (vs @ in-flow (take m cs) vs f)
(run-flow (take m cs) vs s f) f x
⟨*proof*⟩

lemma *sources-member-1*:
assumes
A: ⋀*z a. c = (z ::= a :: com-flow)* ⟹ *z = x* ⟹
y ∈ sources-aux cs′ (vs @ in-flow cs vs f) (run-flow cs vs s f) f x ⟹
sources cs vs s f y ⊆ sources-aux (cs @ cs′) vs s f x
(is ⋀- -. - ⟹ - ⟹ *- ∈ sources-aux - ?vs′ ?s′ - -* ⟹
- ⊆ sources-aux ?cs - - - -)

assumes
B: ⋀*z a b w. c = (z ::= a :: com-flow)* ⟹ *z = x* ⟹
y ∈ sources cs′ ?vs′ ?s′ f w ⟹
sources cs vs s f y ⊆ sources ?cs vs s f w **and**
C: ⋀*z a. c = (z ::= a :: com-flow)* ⟹ *z ≠ x* ⟹
y ∈ sources cs′ ?vs′ ?s′ f x ⟹
sources cs vs s f y ⊆ sources ?cs vs s f x **and**
D: ⋀*z. c = (IN z :: com-flow)* ⟹ *z = x* ⟹
y ∈ sources-aux cs′ ?vs′ ?s′ f x ⟹
sources cs vs s f y ⊆ sources-aux ?cs vs s f x **and**
E: ⋀*z. c = (IN z :: com-flow)* ⟹ *z ≠ x* ⟹
y ∈ sources cs′ ?vs′ ?s′ f x ⟹

sources cs vs s f y ⊆ *sources ?cs vs s f x* **and**
F: ⋀*z*. *c* = (*OUT z* :: *com-flow*) ⟹
y ∈ *sources cs′ ?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources ?cs vs s f x* **and**
G: ⋀*X b w*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources cs′ ?vs′ ?s′ f w* ⟹
sources cs vs s f y ⊆ *sources ?cs vs s f w*
shows *y* ∈ *sources* (*cs′* @ [*c*]) *?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources* (*cs* @ *cs′* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *sources-member-2*:
assumes
A: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹
y ∈ *sources-aux cs′* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
sources cs vs s f y ⊆ *sources-aux* (*cs* @ *cs′*) *vs s f x*
(**is** ⋀- -. - ⟹ - ∈ *sources-aux* - *?vs′ ?s′* - - ⟹
- ⊆ *sources-aux ?cs* - - - -)
assumes
B: ⋀*z*. *c* = (*IN z* :: *com-flow*) ⟹
y ∈ *sources-aux cs′ ?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources-aux ?cs vs s f x* **and**
C: ⋀*z*. *c* = (*OUT z* :: *com-flow*) ⟹
y ∈ *sources-aux cs′ ?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources-aux ?cs vs s f x* **and**
D: ⋀*X*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources-aux cs′ ?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources-aux ?cs vs s f x* **and**
E: ⋀*X b w*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources cs′ ?vs′ ?s′ f w* ⟹
sources cs vs s f y ⊆ *sources ?cs vs s f w*
shows *y* ∈ *sources-aux* (*cs′* @ [*c*]) *?vs′ ?s′ f x* ⟹
sources cs vs s f y ⊆ *sources-aux* (*cs* @ *cs′* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *sources-member*:
y ∈ *sources cs′* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
sources cs vs s f y ⊆ *sources* (*cs* @ *cs′*) *vs s f x*
and *sources-aux-member*:
y ∈ *sources-aux cs′* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
sources cs vs s f y ⊆ *sources-aux* (*cs* @ *cs′*) *vs s f x*
⟨*proof*⟩

lemma *sources-out-member-1*:
assumes
A: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹
y ∈ *sources-out cs′* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
sources cs vs s f y ⊆ *sources-out* (*cs* @ *cs′*) *vs s f x*

(**is** $\bigwedge$- -. - $\Longrightarrow$ - $\in$ *sources-out* - *?vs′ ?s′* - - $\Longrightarrow$
- $\subseteq$ *sources-out ?cs* - - - -)
assumes
B: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow$
$y \in sources\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
sources cs vs s f y $\subseteq$ *sources-out ?cs vs s f x* **and**
C: $\bigwedge z.\ c = (OUT\ z :: com\text{-}flow) \Longrightarrow$
$y \in sources\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
sources cs vs s f y $\subseteq$ *sources-out ?cs vs s f x* **and**
D: $\bigwedge X.\ c = \langle X \rangle \Longrightarrow$
$y \in sources\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
sources cs vs s f y $\subseteq$ *sources-out ?cs vs s f x*
shows $y \in sources\text{-}out\ (cs' @ [c])\ ?vs'\ ?s'\ f\ x \Longrightarrow$
sources cs vs s f y $\subseteq$ *sources-out* $(cs @ cs' @ [c])$ *vs s f x*
⟨*proof*⟩

lemma *sources-out-member*:
$y \in sources\text{-}out\ cs'\ (vs @ in\text{-}flow\ cs\ vs\ f)\ (run\text{-}flow\ cs\ vs\ s\ f)\ f\ x \Longrightarrow$
sources cs vs s f y $\subseteq$ *sources-out* $(cs @ cs')$ *vs s f x*
⟨*proof*⟩

lemma *tags-member-1*:
assumes
A: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$y \in sources\text{-}aux\ cs'\ (vs @ in\text{-}flow\ cs\ vs\ f)\ (run\text{-}flow\ cs\ vs\ s\ f)\ f\ x \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags-aux* $(cs @ cs')$ *vs s f x*
(**is** $\bigwedge$- -. - $\Longrightarrow$ - $\Longrightarrow$ - $\in$ *sources-aux* - *?vs′ ?s′* - - $\Longrightarrow$
- $\subseteq$ *tags-aux ?cs* - - - -)
assumes
B: $\bigwedge z\ a\ b\ w.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$y \in sources\ cs'\ ?vs'\ ?s'\ f\ w \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags ?cs vs s f w* **and**
C: $\bigwedge z\ a.\ c = (z ::= a :: com\text{-}flow) \Longrightarrow z \neq x \Longrightarrow$
$y \in sources\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags ?cs vs s f x* **and**
D: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow z = x \Longrightarrow$
$y \in sources\text{-}aux\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags-aux ?cs vs s f x* **and**
E: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow z \neq x \Longrightarrow$
$y \in sources\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags ?cs vs s f x* **and**
F: $\bigwedge z.\ c = (OUT\ z :: com\text{-}flow) \Longrightarrow$
$y \in sources\ cs'\ ?vs'\ ?s'\ f\ x \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags ?cs vs s f x* **and**
G: $\bigwedge X\ b\ w.\ c = \langle X \rangle \Longrightarrow$
$y \in sources\ cs'\ ?vs'\ ?s'\ f\ w \Longrightarrow$
tags cs vs s f y $\subseteq$ *tags ?cs vs s f w*
shows $y \in sources\ (cs' @ [c])\ ?vs'\ ?s'\ f\ x \Longrightarrow$

tags cs vs s f y ⊆ *tags* (*cs* @ *cs'* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *tags-member-2*:
assumes
A: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹
y ∈ *sources-aux cs'* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
tags cs vs s f y ⊆ *tags-aux* (*cs* @ *cs'*) *vs s f x*
(**is** ⋀- -. - ⟹ - ∈ *sources-aux* - *?vs' ?s'* - - ⟹
- ⊆ *tags-aux ?cs* - - - -)
assumes
B: ⋀*z*. *c* = (*IN z* :: *com-flow*) ⟹
y ∈ *sources-aux cs' ?vs' ?s' f x* ⟹
tags cs vs s f y ⊆ *tags-aux ?cs vs s f x* **and**
C: ⋀*z*. *c* = (*OUT z* :: *com-flow*) ⟹
y ∈ *sources-aux cs' ?vs' ?s' f x* ⟹
tags cs vs s f y ⊆ *tags-aux ?cs vs s f x* **and**
D: ⋀*X*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources-aux cs' ?vs' ?s' f x* ⟹
tags cs vs s f y ⊆ *tags-aux ?cs vs s f x* **and**
E: ⋀*X b w*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources cs' ?vs' ?s' f w* ⟹
tags cs vs s f y ⊆ *tags ?cs vs s f w*
shows *y* ∈ *sources-aux* (*cs'* @ [*c*]) *?vs' ?s' f x* ⟹
tags cs vs s f y ⊆ *tags-aux* (*cs* @ *cs'* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *tags-member*:
y ∈ *sources cs'* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
tags cs vs s f y ⊆ *tags* (*cs* @ *cs'*) *vs s f x*
and *tags-aux-member*:
y ∈ *sources-aux cs'* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
tags cs vs s f y ⊆ *tags-aux* (*cs* @ *cs'*) *vs s f x*
⟨*proof*⟩

lemma *tags-out-member-1*:
assumes
A: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹
y ∈ *sources-out cs'* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
tags cs vs s f y ⊆ *tags-out* (*cs* @ *cs'*) *vs s f x*
(**is** ⋀- -. - ⟹ - ∈ *sources-out* - *?vs' ?s'* - - ⟹
- ⊆ *tags-out ?cs* - - - -)
assumes
B: ⋀*z*. *c* = (*IN z* :: *com-flow*) ⟹
y ∈ *sources-out cs' ?vs' ?s' f x* ⟹
tags cs vs s f y ⊆ *tags-out ?cs vs s f x* **and**
C: ⋀*z*. *c* = (*OUT z* :: *com-flow*) ⟹
y ∈ *sources-out cs' ?vs' ?s' f x* ⟹

tags cs vs s f y ⊆ *tags-out ?cs vs s f x* **and**
D: ⋀*X*. *c* = ⟨*X*⟩ ⟹
y ∈ *sources-out cs′ ?vs′ ?s′ f x* ⟹
tags cs vs s f y ⊆ *tags-out ?cs vs s f x*
shows *y* ∈ *sources-out* (*cs′* @ [*c*]) *?vs′ ?s′ f x* ⟹
tags cs vs s f y ⊆ *tags-out* (*cs* @ *cs′* @ [*c*]) *vs s f x*
⟨*proof*⟩

lemma *tags-out-member*:
y ∈ *sources-out cs′* (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* ⟹
tags cs vs s f y ⊆ *tags-out* (*cs* @ *cs′*) *vs s f x*
⟨*proof*⟩

lemma *tags-suffix-1*:
assumes
A: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹ *z* = *x* ⟹
tags-aux cs′ (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*}
(**is** ⋀- -. - ⟹ - ⟹ *tags-aux* - *?vs′ ?s′* - - = -)
assumes
B: ⋀*z a b y*. *c* = (*z* ::= *a* :: *com-flow*) ⟹ *z* = *x* ⟹
tags cs′ ?vs′ ?s′ f y =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f y*} **and**
C: ⋀*z a*. *c* = (*z* ::= *a* :: *com-flow*) ⟹ *z* ≠ *x* ⟹
tags cs′ ?vs′ ?s′ f x =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f x*} **and**
D: ⋀*z*. *c* = (*IN z* :: *com-flow*) ⟹ *z* = *x* ⟹
tags-aux cs′ ?vs′ ?s′ f x =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*} **and**
E: ⋀*z*. *c* = (*IN z* :: *com-flow*) ⟹ *z* ≠ *x* ⟹
tags cs′ ?vs′ ?s′ f x =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f x*} **and**
F: ⋀*z*. *c* = (*OUT z* :: *com-flow*) ⟹
tags cs′ ?vs′ ?s′ f x =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f x*} **and**
G: ⋀*X b y*. *c* = ⟨*X*⟩ ⟹
tags cs′ ?vs′ ?s′ f y =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f y*}
shows *tags* (*cs′* @ [*c*]) *?vs′ ?s′ f x* =
{*p*. *case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs*. *c* = *IN w*] + *n*)
∈ *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: ⋀*z a. c* = (*z* ::= *a* :: *com-flow*) ⟹
tags-aux cs′ (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*}
(**is** ⋀- -. - ⟹ *tags-aux* - *?vs′ ?s′* - - = -)
assumes
B: ⋀*z. c* = (*IN z* :: *com-flow*) ⟹
tags-aux cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*} **and**
C: ⋀*z. c* = (*OUT z* :: *com-flow*) ⟹
tags-aux cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*} **and**
D: ⋀*X. c* = ⟨*X*⟩ ⟹
tags-aux cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*} **and**
E: ⋀*X b y. c* = ⟨*X*⟩ ⟹
tags cs′ ?vs′ ?s′ f y =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags* (*cs* @ *cs′*) *vs s f y*}
shows *tags-aux* (*cs′* @ [*c*]) *?vs′ ?s′ f x* =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-aux* (*cs* @ *cs′* @ [*c*]) *vs s f x*}
(**is** - = {*p. case p of* (*w*, *n*) ⇒ *?P w n c*})
⟨*proof*⟩

lemma *tags-suffix*:
tags cs′ (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* = {(*w*, *n*).
(*w*, *length* [*c*←*cs. c* = *IN w*] + *n*) ∈ *tags* (*cs* @ *cs′*) *vs s f x*}
and *tags-aux-suffix*:
tags-aux cs′ (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* = {(*w*, *n*).
(*w*, *length* [*c*←*cs. c* = *IN w*] + *n*) ∈ *tags-aux* (*cs* @ *cs′*) *vs s f x*}
⟨*proof*⟩

lemma *tags-out-suffix-1*:
assumes
A: ⋀*z a. c* = (*z* ::= *a* :: *com-flow*) ⟹
tags-out cs′ (*vs* @ *in-flow cs vs f*) (*run-flow cs vs s f*) *f x* =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-out* (*cs* @ *cs′*) *vs s f x*}
(**is** ⋀- -. - ⟹ *tags-out* - *?vs′ ?s′* - - = -)

assumes
B: $\bigwedge z.\ c = (IN\ z :: com\text{-}flow) \Longrightarrow$
tags-out cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-out* (*cs* @ *cs′*) *vs s f x*} **and**
C: $\bigwedge z.\ c = (OUT\ z :: com\text{-}flow) \Longrightarrow$
tags-out cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-out* (*cs* @ *cs′*) *vs s f x*} **and**
D: $\bigwedge X.\ c = \langle X \rangle \Longrightarrow$
tags-out cs′ ?vs′ ?s′ f x =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-out* (*cs* @ *cs′*) *vs s f x*}
shows *tags-out* (*cs′* @ [*c*]) *?vs′ ?s′ f x* =
{*p. case p of* (*w*, *n*) ⇒ (*w*, *length* [*c*←*cs. c* = *IN w*] + *n*)
∈ *tags-out* (*cs* @ *cs′* @ [*c*]) *vs s f x*}
(**is** - = {*p. case p of* (*w*, *n*) ⇒ *?P w n c*})
⟨*proof*⟩

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*←*cs. c* = *IN w*] + *n*) ∈ *tags-out* (*cs* @ *cs′*) *vs s f x*}
⟨*proof*⟩

lemma *sources-aux-rhs*:
assumes
A: *S* ⊆ {*x*. $s_1 = t_1$ (⊆ *sources-aux* (*flow cfs* @ *cs′*) vs_1 s_1 *f x*)}
(**is** - ⊆ {-. - = - (⊆ *sources-aux* (*?cs* @ -) - - - -)})
assumes
B: $f = f'$ (⊆ vs_1, $vs_1{}'$,
$\bigcup$ {*tags-aux* (*?cs* @ *cs′*) vs_1 s_1 *f x* | *x*. *x* ∈ *S*}) **and**
C: (c_1, s_1, *f*, vs_1, ws_1) →∗{*cfs*} (c_2, s_2, *f*, vs_2, ws_2) **and**
D: *ok-flow-aux-2* s_1 s_2 t_1 t_2 *f f′* vs_1 $vs_1{}'$ *?cs*
shows *S* ⊆ {*x*. $s_2 = t_2$ (⊆ *sources-aux cs′* vs_2 s_2 *f x*)}
⟨*proof*⟩

lemma *sources-rhs*:
assumes
A: *S* ⊆ {*x*. $s_1 = t_1$ (⊆ *sources* (*flow cfs* @ *cs′*) vs_1 s_1 *f x*)}
(**is** - ⊆ {-. - = - (⊆ *sources* (*?cs* @ -) - - - -)})
assumes
B: $f = f'$ (⊆ vs_1, $vs_1{}'$,
$\bigcup$ {*tags* (*?cs* @ *cs′*) vs_1 s_1 *f x* | *x*. *x* ∈ *S*}) **and**
C: (c_1, s_1, *f*, vs_1, ws_1) →∗{*cfs*} (c_2, s_2, *f*, vs_2, ws_2) **and**
D: *ok-flow-aux-2* s_1 s_2 t_1 t_2 *f f′* vs_1 $vs_1{}'$ *?cs*
shows *S* ⊆ {*x*. $s_2 = t_2$ (⊆ *sources cs′* vs_2 s_2 *f x*)}
⟨*proof*⟩

lemma *sources-out-rhs*:
assumes
A: $S \subseteq \{x.\ s_1 = t_1\ (\subseteq \mathit{sources\text{-}out}\ (\mathit{flow}\ \mathit{cfs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x)\}$
(**is** - ⊆ {-. - = - (⊆ *sources-out* (*?cs* @ -) - - - -)})
assumes
B: $f = f'\ (\subseteq \mathit{vs}_1, \mathit{vs}_1',$
$\bigcup\ \{\mathit{tags\text{-}out}\ (\mathit{?cs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x \mid x.\ x \in S\})$ **and**
C: $(c_1, s_1, f, \mathit{vs}_1, \mathit{ws}_1) \rightarrow *\{\mathit{cfs}\}\ (c_2, s_2, f, \mathit{vs}_2, \mathit{ws}_2)$ **and**
D: *ok-flow-aux-2* $s_1\ s_2\ t_1\ t_2\ f\ f'\ \mathit{vs}_1\ \mathit{vs}_1'\ \mathit{?cs}$
shows $S \subseteq \{x.\ s_2 = t_2\ (\subseteq \mathit{sources\text{-}out}\ \mathit{cs}'\ \mathit{vs}_2\ s_2\ f\ x)\}$
⟨*proof*⟩

lemma *tags-aux-rhs*:
assumes
A: $S \subseteq \{x.\ s_1 = t_1\ (\subseteq \mathit{sources\text{-}aux}\ (\mathit{flow}\ \mathit{cfs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x)\}$
(**is** - ⊆ {-. - = - (⊆ *sources-aux* (*?cs* @ -) - - - -)})
assumes
B: $f = f'\ (\subseteq \mathit{vs}_1, \mathit{vs}_1',$
$\bigcup\ \{\mathit{tags\text{-}aux}\ (\mathit{?cs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x \mid x.\ x \in S\})$ **and**
C: $(c_1, s_1, f, \mathit{vs}_1, \mathit{ws}_1) \rightarrow *\{\mathit{cfs}\}\ (c_2, s_2, f, \mathit{vs}_2, \mathit{ws}_2)$ **and**
D: $(c_1', t_1, f', \mathit{vs}_1', \mathit{ws}_1') \rightarrow *\ (c_2', t_2, f', \mathit{vs}_2', \mathit{ws}_2')$ **and**
E: *ok-flow-aux-1* $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ \mathit{vs}_1\ \mathit{vs}_1'\ \mathit{vs}_2\ \mathit{vs}_2'\ \mathit{ws}_1'\ \mathit{ws}_2'\ \mathit{?cs}$
shows $f = f'\ (\subseteq \mathit{vs}_2, \mathit{vs}_2', \bigcup\ \{\mathit{tags\text{-}aux}\ \mathit{cs}'\ \mathit{vs}_2\ s_2\ f\ x \mid x.\ x \in S\})$
⟨*proof*⟩

lemma *tags-rhs*:
assumes
A: $S \subseteq \{x.\ s_1 = t_1\ (\subseteq \mathit{sources}\ (\mathit{flow}\ \mathit{cfs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x)\}$
(**is** - ⊆ {-. - = - (⊆ *sources* (*?cs* @ -) - - - -)})
assumes
B: $f = f'\ (\subseteq \mathit{vs}_1, \mathit{vs}_1',$
$\bigcup\ \{\mathit{tags}\ (\mathit{?cs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x \mid x.\ x \in S\})$ **and**
C: $(c_1, s_1, f, \mathit{vs}_1, \mathit{ws}_1) \rightarrow *\{\mathit{cfs}\}\ (c_2, s_2, f, \mathit{vs}_2, \mathit{ws}_2)$ **and**
D: $(c_1', t_1, f', \mathit{vs}_1', \mathit{ws}_1') \rightarrow *\ (c_2', t_2, f', \mathit{vs}_2', \mathit{ws}_2')$ **and**
E: *ok-flow-aux-1* $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ \mathit{vs}_1\ \mathit{vs}_1'\ \mathit{vs}_2\ \mathit{vs}_2'\ \mathit{ws}_1'\ \mathit{ws}_2'\ \mathit{?cs}$
shows $f = f'\ (\subseteq \mathit{vs}_2, \mathit{vs}_2', \bigcup\ \{\mathit{tags}\ \mathit{cs}'\ \mathit{vs}_2\ s_2\ f\ x \mid x.\ x \in S\})$
⟨*proof*⟩

lemma *tags-out-rhs*:
assumes
A: $S \subseteq \{x.\ s_1 = t_1\ (\subseteq \mathit{sources\text{-}out}\ (\mathit{flow}\ \mathit{cfs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x)\}$
(**is** - ⊆ {-. - = - (⊆ *sources-out* (*?cs* @ -) - - - -)})
assumes
B: $f = f'\ (\subseteq \mathit{vs}_1, \mathit{vs}_1',$
$\bigcup\ \{\mathit{tags\text{-}out}\ (\mathit{?cs}\ @\ \mathit{cs}')\ \mathit{vs}_1\ s_1\ f\ x \mid x.\ x \in S\})$ **and**
C: $(c_1, s_1, f, \mathit{vs}_1, \mathit{ws}_1) \rightarrow *\{\mathit{cfs}\}\ (c_2, s_2, f, \mathit{vs}_2, \mathit{ws}_2)$ **and**
D: $(c_1', t_1, f', \mathit{vs}_1', \mathit{ws}_1') \rightarrow *\ (c_2', t_2, f', \mathit{vs}_2', \mathit{ws}_2')$ **and**
E: *ok-flow-aux-1* $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ \mathit{vs}_1\ \mathit{vs}_1'\ \mathit{vs}_2\ \mathit{vs}_2'\ \mathit{ws}_1'\ \mathit{ws}_2'\ \mathit{?cs}$

shows $f = f'\ (\subseteq\ vs_2,\ vs_2',\ \bigcup\ \{tags\text{-}out\ cs'\ vs_2\ s_2\ f\ x \mid x.\ x \in S\})$
⟨*proof*⟩

lemma *ctyping2-term-seq*:
assumes
A: $\bigwedge B\ Y\ p.\ (U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV \Longrightarrow \exists p'.\ (c_1,\ p) \Rightarrow p'$ **and**
B: $\bigwedge q\ B\ Y\ B'\ Y'\ p.\ (U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ q \Longrightarrow (B,\ Y) = q \Longrightarrow$
$(U,\ v) \models c_2\ (\subseteq B,\ Y) = Some\ (B',\ Y') \Longrightarrow$
$\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV \Longrightarrow \exists p'.\ (c_2,\ p) \Rightarrow p'$ **and**
C: $(U,\ v) \models c_1;;\ c_2\ (\subseteq A,\ X) = Some\ (B',\ Y')$ **and**
D: $\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV$
shows $\exists p'.\ (c_1;;\ c_2,\ p) \Rightarrow p'$
⟨*proof*⟩

lemma *ctyping2-term-or*:
assumes
A: $\bigwedge B\ Y\ p.\ (U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV \Longrightarrow \exists p'.\ (c_1,\ p) \Rightarrow p'$ **and**
B: $(U,\ v) \models c_1\ OR\ c_2\ (\subseteq A,\ X) = Some\ (B',\ Y')$ **and**
C: $\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV$
shows $\exists p'.\ (c_1\ OR\ c_2,\ p) \Rightarrow p'$
⟨*proof*⟩

lemma *ctyping2-term-if*:
assumes
A: $\bigwedge U'\ q\ B_1\ B_2\ B\ Y\ p.$
$(U',\ q) = (insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = q \Longrightarrow (U',\ v) \models c_1\ (\subseteq B_1,\ X) = Some\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U'.\ \neg\ C\colon Z \rightsquigarrow UNIV \Longrightarrow \exists p'.\ (c_1,\ p) \Rightarrow p'$ **and**
B: $\bigwedge U'\ q\ B_1\ B_2\ B\ Y\ p.$
$(U',\ q) = (insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = q \Longrightarrow (U',\ v) \models c_2\ (\subseteq B_2,\ X) = Some\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U'.\ \neg\ C\colon Z \rightsquigarrow UNIV \Longrightarrow \exists p'.\ (c_2,\ p) \Rightarrow p'$ **and**
C: $(U,\ v) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A,\ X) = Some\ (B,\ Y)$ **and**
D: $\exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV$
shows $\exists p'.\ (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ p) \Rightarrow p'$
⟨*proof*⟩

lemma *ctyping2-term*:
$[\![(U,\ v) \models c\ (\subseteq A,\ X) = Some\ (B,\ Y);\ \exists (C,\ Z) \in U.\ \neg\ C\colon Z \rightsquigarrow UNIV]\!] \Longrightarrow$
$\exists p'.\ (c,\ p) \Rightarrow p'$
⟨*proof*⟩

lemma *ctyping2-confine-seq*:
assumes
A: $\bigwedge s'\ f'\ vs'\ ws'\ A\ B\ X\ Y\ U\ v.\ p = (s',\ f',\ vs',\ ws') \Longrightarrow$

$(U,\ v) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow \exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S \Longrightarrow$
$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ vs)\ vs'.\ \mathit{fst}\ p \in S] = [] \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ ws)\ ws'.\ \mathit{fst}\ p \in S] = []$
(**is** $\bigwedge s'$ - vs' ws' - - - - - -. - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$
$?P\ s\ s'\ vs\ vs'\ ws\ ws'$)
assumes
$B\colon \bigwedge s'\ f'\ vs'\ ws'\ A\ B\ X\ Y\ U\ v.\ p = (s',\ f',\ vs',\ ws') \Longrightarrow$
$(U,\ v) \models c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow \exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S \Longrightarrow$
$?P\ s'\ s''\ vs'\ vs''\ ws'\ ws''$ **and**
$C\colon (c_1,\ s,\ f,\ vs,\ ws) \Rightarrow p$ **and**
$D\colon (c_2,\ p) \Rightarrow (s'',\ f'',\ vs'',\ ws'')$ **and**
$E\colon (U,\ v) \models c_1;;\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B',\ Y')$ **and**
$F\colon \exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S$
shows $?P\ s\ s''\ vs\ vs''\ ws\ ws''$
⟨*proof*⟩

lemma *ctyping2-confine-or-lhs*:
assumes
$A\colon \bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S \Longrightarrow$
$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ vs)\ vs'.\ \mathit{fst}\ p \in S] = [] \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ ws)\ ws'.\ \mathit{fst}\ p \in S] = []$
(**is** $\bigwedge$- - - - - -. - $\Longrightarrow$ - $\Longrightarrow$ $?P$)
assumes
$B\colon (U,\ v) \models c_1\ \mathit{OR}\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B',\ Y')$ **and**
$C\colon \exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S$
shows $?P$
⟨*proof*⟩

lemma *ctyping2-confine-or-rhs*:
assumes
$A\colon \bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S \Longrightarrow$
$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ vs)\ vs'.\ \mathit{fst}\ p \in S] = [] \wedge$
$[p{\leftarrow}\mathit{drop}\ (\mathit{length}\ ws)\ ws'.\ \mathit{fst}\ p \in S] = []$
(**is** $\bigwedge$- - - - - -. - $\Longrightarrow$ - $\Longrightarrow$ $?P$)
assumes
$B\colon (U,\ v) \models c_1\ \mathit{OR}\ c_2\ (\subseteq A,\ X) = \mathit{Some}\ (B',\ Y')$ **and**
$C\colon \exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S$
shows $?P$
⟨*proof*⟩

lemma *ctyping2-confine-if-true*:
assumes
$A\colon \bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_1\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ C\colon Z \rightsquigarrow\mid S \Longrightarrow$

$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}drop\ (length\ vs)\ vs'.\ fst\ p \in S] = [] \wedge$
$[p{\leftarrow}drop\ (length\ ws)\ ws'.\ fst\ p \in S] = []$
(**is** $\bigwedge$- - - - - -. - $\Longrightarrow$ - $\Longrightarrow$ *?P*)
assumes
B: $(U,\ v) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A,\ X) = Some\ (B,\ Y)$ **and**
C: $\exists (C,\ Z) \in U.\ C{:}\ Z \rightsquigarrow| S$
shows *?P*
⟨*proof*⟩

lemma *ctyping2-confine-if-false*:
assumes
A: $\bigwedge A\ B\ X\ Y\ U\ v.\ (U,\ v) \models c_2\ (\subseteq A,\ X) = Some\ (B,\ Y) \Longrightarrow$
$\exists (C,\ Z) \in U.\ C{:}\ Z \rightsquigarrow| S \Longrightarrow$
$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}drop\ (length\ vs)\ vs'.\ fst\ p \in S] = [] \wedge$
$[p{\leftarrow}drop\ (length\ ws)\ ws'.\ fst\ p \in S] = []$
(**is** $\bigwedge$- - - - - -. - $\Longrightarrow$ - $\Longrightarrow$ *?P*)
assumes
B: $(U,\ v) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A,\ X) = Some\ (B,\ Y)$ **and**
C: $\exists (C,\ Z) \in U.\ C{:}\ Z \rightsquigarrow| S$
shows *?P*
⟨*proof*⟩

lemma *ctyping2-confine*:
$[\![(c,\ s,\ f,\ vs,\ ws) \Rightarrow (s',\ f',\ vs',\ ws');$
$(U,\ v) \models c\ (\subseteq A,\ X) = Some\ (B,\ Y);\ \exists (C,\ Z) \in U.\ C{:}\ Z \rightsquigarrow| S]\!] \Longrightarrow$
$s = s'\ (\subseteq S) \wedge$
$[p{\leftarrow}drop\ (length\ vs)\ vs'.\ fst\ p \in S] = [] \wedge$
$[p{\leftarrow}drop\ (length\ ws)\ ws'.\ fst\ p \in S] = []$
(**is** $[\![$-; -; -$]\!] \Longrightarrow$ *?P s s' vs vs' ws ws'*)
⟨*proof*⟩

lemma *eq-states-assign*:
assumes
A: $S \subseteq \{y.\ s = t\ (\subseteq sources\ [x ::= a]\ vs\ s\ f\ y)\}$ **and**
B: $x \in S$ **and**
C: $s \in Univ\ A\ (\subseteq state \cap X)$ **and**
D: $Univ?\ A\ X{:}\ avars\ a \rightsquigarrow \{x\}$
shows $s = t\ (\subseteq avars\ a)$
⟨*proof*⟩

lemma *eq-states-while*:
assumes
A: $S \subseteq \{x.\ s = t\ (\subseteq sources\text{-}aux\ (\langle bvars\ b\rangle\ \#\ cs)\ vs\ s\ f\ x)\}$ **and**
B: $S \neq \{\}$ **and**
C: $s \in Univ\ A\ (\subseteq state \cap X) \cup Univ\ C\ (\subseteq state \cap Y)$ **and**
D: $Univ?\ A\ X \cup Univ?\ C\ Y{:}\ bvars\ b \rightsquigarrow UNIV$

shows $s = t\ (\subseteq bvars\ b)$
⟨*proof*⟩

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: $(\{\},\ False) \models c\ (\subseteq B_1,\ X) = Some\ (D,\ Z)$ **and**
F: $(\{\},\ False) \models c\ (\subseteq B_1{'},\ Y) = Some\ (D',\ Z')$ **and**
G: $bval\ b\ s$
shows $s \in Univ\ A\ (\subseteq state \cap X) \cup Univ\ C\ (\subseteq state \cap Y) \Longrightarrow$
$s' \in Univ\ A\ (\subseteq state \cap X) \cup Univ\ C\ (\subseteq state \cap Y)$
⟨*proof*⟩

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 *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*!]:
$[\![(SKIP,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1);$
$(c_1,\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c_2,\ s_2,\ f,\ vs_2,\ ws_2)]\!] \Longrightarrow$
ok-flow-aux U $c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2$ (*flow* cfs_2)

⟨*proof*⟩

lemma *ctyping2-correct-aux-assign*:
assumes
A: $(U,\ v) \models x ::= a\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
B: $s \in Univ\ A\ (\subseteq state \cap X)$ **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* $c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-input*:
assumes
A: $(U,\ v) \models IN\ x\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
B: $(IN\ x,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1)$ **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* $c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-output*:
assumes
A: $(U,\ v) \models OUT\ x\ (\subseteq A,\ X) = Some\ (B,\ Y)$ **and**
B: $(OUT\ x,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1)$ **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* $c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-seq*:
assumes
A: $(U,\ v) \models c_1;;\ c_2\ (\subseteq A,\ X) = Some\ (C,\ Z)$ **and**
B: $\bigwedge B\ Y\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ (B,\ Y) \Longrightarrow$
$s \in Univ\ A\ (\subseteq state \cap X) \Longrightarrow$
$(c_1,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux U $c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
C: $\bigwedge p\ B\ Y\ C\ Z\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ p \Longrightarrow (B,\ Y) = p \Longrightarrow$
$(U,\ v) \models c_2\ (\subseteq B,\ Y) = Some\ (C,\ Z) \Longrightarrow$
$s \in Univ\ B\ (\subseteq state \cap Y) \Longrightarrow$
$(c_2,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux U $c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
D: $s \in Univ\ A\ (\subseteq state \cap X)$ **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* $c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-or*:
assumes
A: $(U,\ v) \models c_1\ OR\ c_2\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
B: $\bigwedge C\ Y\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U,\ v) \models c_1\ (\subseteq A,\ X) = Some\ (C,\ Y) \Longrightarrow$
$s \in Univ\ A\ (\subseteq state \cap X) \Longrightarrow$
$(c_1,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux $U\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
C: $\bigwedge C\ Y\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U,\ v) \models c_2\ (\subseteq A,\ X) = Some\ (C,\ Y) \Longrightarrow$
$s \in Univ\ A\ (\subseteq state \cap X) \Longrightarrow$
$(c_2,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux $U\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
D: $s \in Univ\ A\ (\subseteq state \cap X)$ **and**
E: $(c_1\ OR\ c_2,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1)$ **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* $U\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-if*:
assumes
A: $(U,\ v) \models IF\ b\ THEN\ c_1\ ELSE\ c_2\ (\subseteq A,\ X) = Some\ (C,\ Y)$ **and**
B: $\bigwedge U'\ p\ B_1\ B_2\ C_1\ Y_1\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U',\ p) = (insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = p \Longrightarrow$
$(U',\ v) \models c_1\ (\subseteq B_1,\ X) = Some\ (C_1,\ Y_1) \Longrightarrow$
$s \in Univ\ B_1\ (\subseteq state \cap X) \Longrightarrow$
$(c_1,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux $U'\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
C: $\bigwedge U'\ p\ B_1\ B_2\ C_2\ Y_2\ c'\ c''\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(U',\ p) = (insert\ (Univ?\ A\ X,\ bvars\ b)\ U,\ \models b\ (\subseteq A,\ X)) \Longrightarrow$
$(B_1,\ B_2) = p \Longrightarrow$
$(U',\ v) \models c_2\ (\subseteq B_2,\ X) = Some\ (C_2,\ Y_2) \Longrightarrow$
$s \in Univ\ B_2\ (\subseteq state \cap X) \Longrightarrow$
$(c_2,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c',\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c',\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c'',\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
ok-flow-aux $U'\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$ **and**
D: $s \in Univ\ A\ (\subseteq state \cap X)$ **and**
E: $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}$
$(c',\ s_1,\ f,\ vs_1,\ ws_1)$ **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* $U\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux-while*:
assumes

A: $(U,\ v) \models \mathit{WHILE}\ b\ \mathit{DO}\ c\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ W)$ **and**
B: $\bigwedge B_1\ B_2\ C\ Y\ B_1{'}\ B_2{'}\ D\ Z\ c_1\ c_2\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(B_1,\ B_2) = \models b\ (\subseteq A,\ X) \Longrightarrow$
$(C,\ Y) = \vdash c\ (\subseteq B_1,\ X) \Longrightarrow$
$(B_1{'},\ B_2{'}) = \models b\ (\subseteq C,\ Y) \Longrightarrow$
$\forall (B,\ W) \in \mathit{insert}\ (\mathit{Univ?}\ A\ X \cup \mathit{Univ?}\ C\ Y,\ \mathit{bvars}\ b)\ U.$
B: $W \rightsquigarrow \mathit{UNIV} \Longrightarrow$
$(\{\},\ \mathit{False}) \models c\ (\subseteq B_1,\ X) = \mathit{Some}\ (D,\ Z) \Longrightarrow$
$s \in \mathit{Univ}\ B_1\ (\subseteq \mathit{state} \cap X) \Longrightarrow$
$(c,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c_1,\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c_2,\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
$\mathit{ok\text{-}flow\text{-}aux}\ \{\}\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (\mathit{flow}\ cfs_2)$ **and**
C: $\bigwedge B_1\ B_2\ C\ Y\ B_1{'}\ B_2{'}\ D{'}\ Z{'}\ c_1\ c_2\ s\ s_1\ s_2\ vs_0\ vs_1\ vs_2\ ws_0\ ws_1\ ws_2\ cfs_1\ cfs_2.$
$(B_1,\ B_2) = \models b\ (\subseteq A,\ X) \Longrightarrow$
$(C,\ Y) = \vdash c\ (\subseteq B_1,\ X) \Longrightarrow$
$(B_1{'},\ B_2{'}) = \models b\ (\subseteq C,\ Y) \Longrightarrow$
$\forall (B,\ W) \in \mathit{insert}\ (\mathit{Univ?}\ A\ X \cup \mathit{Univ?}\ C\ Y,\ \mathit{bvars}\ b)\ U.$
B: $W \rightsquigarrow \mathit{UNIV} \Longrightarrow$
$(\{\},\ \mathit{False}) \models c\ (\subseteq B_1{'},\ Y) = \mathit{Some}\ (D{'},\ Z{'}) \Longrightarrow$
$s \in \mathit{Univ}\ B_1{'}\ (\subseteq \mathit{state} \cap Y) \Longrightarrow$
$(c,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1) \Longrightarrow$
$(c_1,\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c_2,\ s_2,\ f,\ vs_2,\ ws_2) \Longrightarrow$
$\mathit{ok\text{-}flow\text{-}aux}\ \{\}\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (\mathit{flow}\ cfs_2)$ **and**
D: $s \in \mathit{Univ}\ A\ (\subseteq \mathit{state} \cap X)$ **and**
E: $(\mathit{WHILE}\ b\ \mathit{DO}\ c,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1)$ **and**
F: $(c_1,\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c_2,\ s_2,\ f,\ vs_2,\ ws_2)$
shows $\mathit{ok\text{-}flow\text{-}aux}\ U\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (\mathit{flow}\ cfs_2)$
⟨*proof*⟩

lemma *ctyping2-correct-aux*:
$[\![(U,\ v) \models c\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y);\ s \in \mathit{Univ}\ A\ (\subseteq \mathit{state} \cap X);$
$(c,\ s,\ f,\ vs_0,\ ws_0) \rightarrow* \{cfs_1\}\ (c_1,\ s_1,\ f,\ vs_1,\ ws_1);$
$(c_1,\ s_1,\ f,\ vs_1,\ ws_1) \rightarrow* \{cfs_2\}\ (c_2,\ s_2,\ f,\ vs_2,\ ws_2)]\!] \Longrightarrow$
$\mathit{ok\text{-}flow\text{-}aux}\ U\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (\mathit{flow}\ cfs_2)$
⟨*proof*⟩

theorem *ctyping2-correct*:
assumes A: $(U,\ v) \models c\ (\subseteq A,\ X) = \mathit{Some}\ (B,\ Y)$
shows $\mathit{correct}\ c\ A\ X$
⟨*proof*⟩

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.

[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.

[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, 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.