

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

Pasquale Noce

Senior Staff Firmware Engineer at HID Global, Italy

pasquale dot noce dot lavoro at gmail dot com

pasquale dot noce at hidglobal dot com

February 6, 2026

Abstract

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

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

Contents

1	Extension of language IMP with inputs, outputs, and non-determinism	2
----------	--	----------

1.1	Extended syntax	3
1.2	Extended big-step semantics	4
1.3	Extended small-step semantics	6
1.4	Equivalence of big-step and small-step semantics	7
2	Underlying concepts and formal definitions	9
2.1	Global context definitions	9
2.2	Local context definitions	12
3	Idempotence of the auxiliary type system meant for loop bodies	25
3.1	Local context proofs	25
4	Overapproximation of program semantics by the main type system	48
4.1	Global context proofs	49
4.2	Local context proofs	49
5	Sufficiency of well-typedness for information flow correctness: propaedeutic lemmas	80
5.1	Global context proofs	80
5.2	Local context proofs	93
6	Sufficiency of well-typedness for information flow correctness: main theorem	145
6.1	Local context proofs	146

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

```

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

```

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

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

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

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

1.1 Extended syntax

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

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

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

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

1.2 Extended big-step semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3 Extended small-step semantics

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

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

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

Assign:

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

Input:

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

Output:

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

Seq1:

(*SKIP*;; *c*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*) → *cf*

inductive-cases *seqE* [elim!]: (*c*₁;; *c*₂, *p*) → *cf*

inductive-cases *orE* [elim!]: (*c*₁ *OR* *c*₂, *p*) → *cf*

inductive-cases *ifE* [elim!]: (*IF* *b* *THEN* *c*₁ *ELSE* *c*₂, *p*) → *cf*

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

abbreviation *small-steps* :: *com* × *stage* ⇒ *com* × *stage* ⇒ *bool*
 (**infix** <→* > 55) **where**
cf →* *cf'* ≡ *star small-step cf cf'*

function *small-steps1* ::
com × *stage* ⇒ (*com* × *stage*) *list* ⇒ *com* × *stage* ⇒ *bool*
 (<(- →* '{-}' -) > [51, 51] 55)
where
cf →*{[]} *cf'* = (*cf* = *cf'*) |
cf →*{*cfs* @ [*cf'*]} *cf''* = (*cf* →*{*cfs*} *cf'* ∧ *cf'* → *cf''*)

by (*atomize-elim*, *auto intro: rev-cases*)
termination by *lexicographic-order*

1.4 Equivalence of big-step and small-step semantics

lemma *star-seq2*:
 (*c*₁, *p*) →* (*c*₁', *p'*) ⇒ (*c*₁;; *c*₂, *p*) →* (*c*₁';; *c*₂, *p'*)
proof (*induction rule: star-induct*)
case *refl*
thus ?*case*
by *simp*
next
case *step*
thus ?*case*
by (*blast intro: star.step*)
qed

lemma *seq-comp*:
 [(*c*₁, *p*₁) →* (*SKIP*, *p*₂); (*c*₂, *p*₂) →* (*SKIP*, *p*₃)] ⇒
 (*c*₁;; *c*₂, *p*₁) →* (*SKIP*, *p*₃)
by (*blast intro: star.step star-seq2 star-trans*)

lemma *big-to-small*:
cf ⇒ *p* ⇒ *cf* →* (*SKIP*, *p*)
proof (*induction rule: big-step.induct*)
fix *c*₁ *c*₂ *p*₁ *p*₂ *p*₃
assume (*c*₁, *p*₁) →* (*SKIP*, *p*₂) **and** (*c*₂, *p*₂) →* (*SKIP*, *p*₃)

```

thus ( $c_1;; c_2, p_1$ )  $\rightarrow^*$  (SKIP,  $p_3$ )
  by (rule seq-comp)
next
  fix  $c_1 c_2 p p'$ 
  assume ( $c_1, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
  thus ( $c_1$  OR  $c_2, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
    by (blast intro: star.step)
next
  fix  $c_1 c_2 p p'$ 
  assume ( $c_2, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
  thus ( $c_1$  OR  $c_2, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
    by (blast intro: star.step)
next
  fix  $b c_1 c_2 s p p'$ 
  assume bval  $b s$ 
  hence (IF  $b$  THEN  $c_1$  ELSE  $c_2, s, p$ )  $\rightarrow$  ( $c_1, s, p$ )
    by simp
  moreover assume ( $c_1, s, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
  ultimately show
    (IF  $b$  THEN  $c_1$  ELSE  $c_2, s, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
    by (simp add: star.step)
next
  fix  $b c_1 c_2 s p p'$ 
  assume  $\neg$  bval  $b s$ 
  hence (IF  $b$  THEN  $c_1$  ELSE  $c_2, s, p$ )  $\rightarrow$  ( $c_2, s, p$ )
    by simp
  moreover assume ( $c_2, s, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
  ultimately show
    (IF  $b$  THEN  $c_1$  ELSE  $c_2, s, p$ )  $\rightarrow^*$  (SKIP,  $p'$ )
    by (simp add: star.step)
next
  fix  $b c s_1 s_2 s_3 p_1 p_2 p_3$ 
  assume bval  $b s_1$ 
  hence (WHILE  $b$  DO  $c, s_1, p_1$ )  $\rightarrow^*$  ( $c;;$  WHILE  $b$  DO  $c, s_1, p_1$ )
    by simp
  moreover assume
    ( $c, s_1, p_1$ )  $\rightarrow^*$  (SKIP,  $s_2, p_2$ ) and
    (WHILE  $b$  DO  $c, s_2, p_2$ )  $\rightarrow^*$  (SKIP,  $s_3, p_3$ )
  hence ( $c;;$  WHILE  $b$  DO  $c, s_1, p_1$ )  $\rightarrow^*$  (SKIP,  $s_3, p_3$ )
    by (rule seq-comp)
  ultimately show (WHILE  $b$  DO  $c, s_1, p_1$ )  $\rightarrow^*$  (SKIP,  $s_3, p_3$ )
    by (blast intro: star-trans)
qed fastforce+

```

lemma *small1-big-continue*:

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

by (*induction arbitrary: p rule: small-step.induct, force+*)

lemma *small-to-big*:

```

cf →* (SKIP, p) ⇒ cf ⇒ p
by (induction cf (SKIP, p) rule: star.induct,
auto intro: small1-big-continue)

```

```

lemma big-iff-small:
cf ⇒ p = cf →* (SKIP, p)
by (blast intro: big-to-small small-to-big)

```

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 $state\text{-}upd = vname \times val\ option$

definition $eq\text{-}streams ::$

$stream \Rightarrow stream \Rightarrow inputs \Rightarrow inputs \Rightarrow tag\ set \Rightarrow bool$
 $\langle(- = - '(\subseteq -, -, -') \rangle [51, 51] 50) \mathbf{where}$
 $f = f' (\subseteq vs, vs', T) \equiv \forall (x, n) \in T.$
 $f\ x (length [p \leftarrow vs. fst\ p = x] + n) =$
 $f'\ x (length [p \leftarrow vs'. fst\ p = x] + n)$

abbreviation $eq\text{-}states :: state \Rightarrow state \Rightarrow vname\ set \Rightarrow bool$

$\langle(- = - '(\subseteq -') \rangle [51, 51] 50) \mathbf{where}$
 $s = t (\subseteq X) \equiv \forall x \in X. s\ x = t\ x$

abbreviation $univ\text{-}states :: state\ set \Rightarrow vname\ set \Rightarrow state\ set$

$\langle(Univ - '(\subseteq -') \rangle [51] 75) \mathbf{where}$
 $Univ\ A (\subseteq X) \equiv \{s. \exists t \in A. s = t (\subseteq X)\}$

abbreviation $univ\text{-}vars\text{-}if :: state\ set \Rightarrow vname\ set \Rightarrow vname\ set$

$\langle(Univ?? - -) \rangle [51, 75] 75) \mathbf{where}$
 $Univ??\ A\ X \equiv \text{if } A = \{\} \text{ then } UNIV \text{ else } X$

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

primrec $avars :: aexp \Rightarrow vname\ set \mathbf{where}$

$avars (N\ i) = \{\}$ |
 $avars (V\ x) = \{x\}$ |
 $avars (Plus\ a_1\ a_2) = avars\ a_1 \cup avars\ a_2$

primrec $bvars :: bexp \Rightarrow vname\ set \mathbf{where}$

$bvars (Bc\ v) = \{\}$ |
 $bvars (Not\ b) = bvars\ b$ |
 $bvars (And\ b_1\ b_2) = bvars\ b_1 \cup bvars\ b_2$ |
 $bvars (Less\ a_1\ a_2) = avars\ a_1 \cup avars\ a_2$

fun $no\text{-}upd :: flow \Rightarrow vname\ set \Rightarrow bool \mathbf{where}$

$no\text{-}upd (x ::= - \# cs)\ X = (x \notin X \wedge no\text{-}upd\ cs\ X) |$
 $no\text{-}upd (IN\ x \# cs)\ X = (x \notin X \wedge no\text{-}upd\ cs\ X) |$
 $no\text{-}upd (OUT\ x \# cs)\ X = (x \notin X \wedge no\text{-}upd\ cs\ X) |$
 $no\text{-}upd (- \# cs)\ X = no\text{-}upd\ cs\ X |$
 $no\text{-}upd - = True$

fun $flow\text{-}aux :: com\ list \Rightarrow flow \mathbf{where}$

$flow\text{-}aux (x ::= a \# cs) = (x ::= a) \# flow\text{-}aux\ cs |$
 $flow\text{-}aux (IN\ x \# cs) = IN\ x \# flow\text{-}aux\ cs |$
 $flow\text{-}aux (OUT\ x \# cs) = OUT\ x \# flow\text{-}aux\ cs |$
 $flow\text{-}aux (IF\ b\ THEN - ELSE - \# cs) = \langle bvars\ b \rangle \# flow\text{-}aux\ cs |$

$flow_aux (WHILE\ b\ DO\ -\ \# cs) = \langle bvars\ b \rangle \# flow_aux\ cs \mid$
 $flow_aux (c;;\ -\ \# cs) = flow_aux (c\ \# cs) \mid$
 $flow_aux (-\ \# cs) = flow_aux\ cs \mid$
 $flow_aux [] = []$

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

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

proof *atomize-elim*

fix $p :: flow \times inputs \times stream$

show

$(\exists cs\ x\ a\ vs\ f.\ p = (cs\ @\ [x\ ::= a],\ vs,\ f)) \vee$
 $(\exists cs\ x\ vs\ f.\ p = (cs\ @\ [IN\ x],\ vs,\ f)) \vee$
 $(\exists cs\ x\ vs\ f.\ p = (cs\ @\ [OUT\ x],\ vs,\ f)) \vee$
 $(\exists cs\ X\ vs\ f.\ p = (cs\ @\ [X],\ vs,\ f)) \vee$
 $(\exists vs\ f.\ p = ([],\ vs,\ f))$
by (*cases p, metis com-flow.exhaust rev-exhaust*)

qed *auto*

termination by *lexicographic-order*

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

proof *atomize-elim*

fix $p :: flow \times inputs \times state \times stream$

show

$(\exists cs\ x\ a\ vs\ s\ f.\ p = (cs\ @\ [x\ ::= a],\ vs,\ s,\ f)) \vee$
 $(\exists cs\ x\ vs\ s\ f.\ p = (cs\ @\ [IN\ x],\ vs,\ s,\ f)) \vee$
 $(\exists cs\ x\ vs\ s\ f.\ p = (cs\ @\ [OUT\ x],\ vs,\ s,\ f)) \vee$
 $(\exists cs\ X\ vs\ s\ f.\ p = (cs\ @\ [X],\ vs,\ s,\ f)) \vee$

```

    (∃ vs s f. p = ([], vs, s, f))
  by (cases p, metis com-flow.exhaust rev-exhaust)
qed auto

```

termination by lexicographic-order

```

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 atomize-elim

```

fix p :: flow × inputs × state × stream
show
  (∃ cs x a vs s f. p = (cs @ [x ::= a], vs, s, f)) ∨
  (∃ cs x vs s f. p = (cs @ [IN x], vs, s, f)) ∨
  (∃ cs x vs s f. p = (cs @ [OUT x], vs, s, f)) ∨
  (∃ cs X vs s f. p = (cs @ [⟨X⟩], vs, s, f)) ∨
  (∃ vs s f. p = ([], vs, s, f))
  by (cases p, metis com-flow.exhaust rev-exhaust)
qed auto

```

termination by lexicographic-order

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 $cs_{n+1}, cs_n @ [c_n], \dots, cs_1 @ [c_1]$ of cs and a sequence of variables y_{n+1}, \dots, y_1 , where $n \geq 1$, $cs_{n+1} = cs$, $y_{n+1} = x$, and $y_1 = y$, satisfying the following conditions.

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

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

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

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

function

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

where

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

```

    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 *atomize-elim*

```

fix a :: flow × inputs × state × stream × vname +
  flow × inputs × state × stream × vname

```

show

```

(∃ cs c vs s f x. a = Inl (cs @ [c], vs, s, f, x)) ∨
(∃ vs s f x. a = Inl ([], vs, s, f, x)) ∨
(∃ cs c vs s f x. a = Inr (cs @ [c], vs, s, f, x)) ∨
(∃ vs s f x. a = Inr ([], vs, s, f, x))
by (metis obj-sumE prod-cases3 rev-exhaust)

```

qed *auto*

termination by *lexicographic-order*

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 *⟨X_i⟩* such that:

- $x_i \in X_i$ and

- *run-flow* cs_i *vs* s f : $\text{dom } x_i \rightsquigarrow \text{dom } x$.

function

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

where

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

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

by (*atomize-elim*, *auto intro: rev-cases*)

termination by *lexicographic-order*

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:

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

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

- $y_i \in X_i$ and
 - *run-flow* cs_i *vs* $s f$: $dom y_i \rightsquigarrow dom y_{i+1}$.
- The instruction c_1 is the m -th input instruction *IN* y within flow cs .
 - The right sublist of cs_2 complementary to $cs_1 @ [c_1]$ does not comprise any assignment or input instruction setting variable y (as the input instruction c_1 would otherwise be irrelevant).

function

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

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

where

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

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

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

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

proof *atomize-elim*

fix $a :: flow \times inputs \times state \times stream \times vname +$
 $flow \times inputs \times state \times stream \times vname$

show

$(\exists cs\ c\ vs\ s\ f\ x. a = Inl\ (cs @ [c], vs, s, f, x)) \vee$
 $(\exists vs\ s\ f\ x. a = Inl\ ([], vs, s, f, x)) \vee$
 $(\exists cs\ c\ vs\ s\ f\ x. a = Inr\ (cs @ [c], vs, s, f, x)) \vee$
 $(\exists vs\ s\ f\ x. a = Inr\ ([], vs, s, f, x))$

by (*metis obj-sumE prod-cases3 rev-exhaust*)
qed *auto*

termination by *lexicographic-order*

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*: $\text{dom } x_i \rightsquigarrow \text{dom } x$.

function

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

where

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

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

by (*atomize-elim, auto intro: rev-cases*)

termination by *lexicographic-order*

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

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

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

Furthermore, let:

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

and assume that the following conditions hold.

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

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

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

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

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

abbreviation *ok-flow-1* where

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

abbreviation *ok-flow-2* where

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

abbreviation *ok-flow* where

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

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

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

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

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

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

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

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

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

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

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

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

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

where

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

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* \times *val option*, where the first component is the state variable being updated, and the latter one matches *Some i* or *None* depending on whether its new value can be evaluated to an integer *i* at compile time or not.

Apart from the aforesaid type change, the equations for the constructs in-

cluded 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* \Rightarrow *bool option* ($\langle \vdash - \rangle$ [51] 55) **where**

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

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

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

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

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

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

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

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

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

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

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

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

definition *ctyping1-merge-append* ::
state-upd list set \Rightarrow *state-upd list set* \Rightarrow *state-upd list set*
 (infixl $\langle \sqcup @ \rangle$ 55) **where**
 $A \sqcup @ B \equiv (\text{if } card \text{ } B = \text{Suc } 0 \text{ then } A \text{ else } A \sqcup B) @ B$

primrec *ctyping1-aux* :: *com* \Rightarrow *state-upd list set*
 ($\langle \vdash - \rangle$ [51] 60) **where**

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

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

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

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

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

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

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

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

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

($\langle \vdash - '(\subseteq -, -)' \rangle$ [51] 55) **where**
 $\vdash c (\subseteq A, X) \equiv \text{let } F = \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in \vdash c\}$ in
 $(\{\lambda x. \text{if } f x = \square$
 then $s x$ *else case* $\text{snd} (\text{last } (f x))$ *of* $\text{None} \Rightarrow t x \mid \text{Some } i \Rightarrow i \mid$
 $f s t. f \in F \wedge s \in A\}$,
Univ??? $A \{x. \forall f \in F. \text{if } f x = \square$
 then $x \in X$ *else* $\text{snd} (\text{last } (f x)) \neq \text{None}\}$)

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

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

primrec *btyping2-aux* :: *bexp* \Rightarrow *state set* \Rightarrow *vname set* \Rightarrow *state set option*

($\langle \models - '(\subseteq -, -)' \rangle$ [51] 55) **where**

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

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

$\models \text{And } b_1 b_2 (\subseteq A, X) = (\text{case } (\models b_1 (\subseteq A, X), \models b_2 (\subseteq A, X)) \text{ of}$

$(Some\ B_1, Some\ B_2) \Rightarrow Some\ (B_1 \cap B_2) \mid - \Rightarrow None \mid$

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

definition *btyping2* ::

$bexp \Rightarrow state\ set \Rightarrow vname\ set \Rightarrow state\ set \times state\ set$
 $(\langle \models - \ '(\subseteq -, -)' \rangle [51, 55])$ **where**
 $\models b\ (\subseteq\ A, X) \equiv case\ \models b\ (\subseteq\ A, X)$ of
 $Some\ A' \Rightarrow (A', A - A') \mid - \Rightarrow (A, A)$

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

$(\langle (- : - \rightsquigarrow -) \rangle [51, 51, 51] 50)$ **where**
 $A: X \rightsquigarrow Y \equiv \forall s \in A. \forall x \in X. \forall y \in Y. s: dom\ x \rightsquigarrow dom\ y$

abbreviation *atyping* :: $bool \Rightarrow aexp \Rightarrow vname\ set \Rightarrow bool$

$(\langle (- \models - \ '(\subseteq -)' \rangle [51, 51] 50)$ **where**
 $v \models a\ (\subseteq\ X) \equiv avars\ a = \{\} \vee avars\ a \subseteq state \cap X \wedge v$

definition *univ-states-if* :: $state\ set \Rightarrow vname\ set \Rightarrow state\ set$

$(\langle (Univ? - -) \rangle [51, 75] 75)$ **where**
 $Univ? A\ X \equiv if\ state \subseteq X\ then\ A\ else\ Univ\ A\ (\subseteq\ \{\})$

fun *ctyping2* :: $scope \Rightarrow com \Rightarrow state\ set \Rightarrow vname\ set \Rightarrow config\ option$

$(\langle (- \models - \ '(\subseteq -, -)' \rangle [51, 51] 55)$ **where**

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

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

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

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

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

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**

$ctyping1\text{-idem-lhs } s \ t \ t' \ ys \ ys' \ x \equiv$
 if $[y \leftarrow ys'. \text{fst } y = x] = []$
 then if $[y \leftarrow ys. \text{fst } y = x] = []$
 then $s \ x$
 else case $\text{snd } (\text{last } [y \leftarrow ys. \text{fst } y = x])$ of $\text{None} \Rightarrow t \ x \mid \text{Some } i \Rightarrow i$
 else case $\text{snd } (\text{last } [y \leftarrow ys'. \text{fst } y = x])$ of $\text{None} \Rightarrow t' \ x \mid \text{Some } i \Rightarrow i$

abbreviation $ctyping1\text{-idem-rhs}$ where

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

abbreviation $ctyping1\text{-idem-pred}$ where

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

lemma $ctyping1\text{-merge-aux-no-nil}$:

$ws \in A \sqcup B \Longrightarrow ws \neq []$
by (*erule* $ctyping1\text{-merge-aux.cases}$, *simp-all*)

lemma $ctyping1\text{-merge-aux-empty-lhs}$:

$\{\} \sqcup B = \{[(ys, \text{False})] \mid ys. ys \in B\}$
by (*rule* equalityI , *clarify*, *erule* $ctyping1\text{-merge-aux.induct}$, *auto*)

lemma $ctyping1\text{-merge-aux-empty-rhs}$:

$A \sqcup \{\} = \{[(xs, \text{True})] \mid xs. xs \in A\}$
by (*rule* equalityI , *clarify*, *erule* $ctyping1\text{-merge-aux.induct}$, *auto*)

lemma $ctyping1\text{-merge-empty-lhs}$:

$\{\} \sqcup B = B$
by (*force* *simp*: $ctyping1\text{-merge-def}$ $ctyping1\text{-merge-aux-empty-lhs}$)

lemma $ctyping1\text{-merge-empty-rhs}$:

$A \sqcup \{\} = A$
by (*force* *simp*: $ctyping1\text{-merge-def}$ $ctyping1\text{-merge-aux-empty-rhs}$)

lemma $ctyping1\text{-aux-nonempty}$:

$\vdash c \neq \{\}$
by (*induction* c , *auto* *simp*: *Let-def* $ctyping1\text{-merge-def}$
 $ctyping1\text{-merge-append-def}$ $ctyping1\text{-append-def}$, *fastforce*)

lemma $ctyping1\text{-merge-idem-fst}$:

assumes

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

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

proof –

obtain ws **where** $ws \in \vdash c_1 \sqcup \vdash c_2$ **and** $ys = concat\ (map\ fst\ ws)$
using D **by** $(auto\ simp:\ ctyping1-merge-def)$

thus $?thesis$

proof $(induction\ ws\ arbitrary:\ ys\ rule:\ list.induct,$
 $blast\ dest:\ ctyping1-merge-aux-no-nil)$

fix $w\ ws\ ys$

assume

$F:$ $\bigwedge xs.\ ws \in \vdash c_1 \sqcup \vdash c_2 \implies xs = concat\ (map\ fst\ ws) \implies$
 $ctyping1-idem-pred\ s\ t\ t'\ xs\ ys'\ A\ (\vdash\ c_1 \sqcup \vdash c_2)$ **and**

$G:$ $w \# ws \in \vdash c_1 \sqcup \vdash c_2$

assume $ys = concat\ (map\ fst\ (w \# ws))$

hence $H:$ $ys = fst\ w\ @\ concat\ (map\ fst\ ws)$

$(is\ ys = ?x\ @\ ?xs)$

by $simp$

have $ctyping1-idem-pred\ s\ t\ t'\ ?xs\ ys'\ A\ (\vdash\ c_1 \sqcup \vdash c_2)$

proof $(cases\ ws)$

case Nil

show $?thesis$

apply $(rule\ exI\ [of\ -\ \lambda x.\ [y \leftarrow ys'.\ fst\ y = x]])$

apply $(rule\ exI\ [of\ -\ s])$

apply $(rule\ conjI)$

apply $(rule\ exI\ [of\ -\ t'])$

by $(auto\ simp:\ C\ E\ Nil)$

next

case $Cons$

have $ws \in \vdash c_1 \sqcup \vdash c_2$

using G **by** $(rule\ ctyping1-merge-aux.cases,\ simp-all\ add:\ Cons)$

thus $?thesis$

using F **by** $simp$

qed

then obtain f **and** s' **and** t'' **and** ys'' **where**

$I:$ $ctyping1-idem-lhs\ s\ t\ t'\ ?xs\ ys' =$

$ctyping1-idem-rhs\ f\ s'\ t''$ **and**

$J:$ $\forall x.\ (f\ x = [] \iff [y \leftarrow ?xs\ @\ ys'.\ fst\ y = x] = []) \wedge$

$(f\ x \neq [] \implies last\ (f\ x) = last\ [y \leftarrow ?xs\ @\ ys'.\ fst\ y = x])$ **and**

$K:$ $f = (\lambda x.\ [y \leftarrow ys''.\ fst\ y = x])$ **and**

$L:$ $ys'' \in \vdash c_1 \sqcup \vdash c_2$ **and**

$M:$ $s' \in A$

by $auto$

obtain ws'' **where**

```

N: ws'' ∈ ⊢ c₁ ⊔ ⊢ c₂ and
O: ys'' = concat (map fst ws'')
using L by (auto simp: ctying1-merge-def)
show ctying1-idem-pred s t t' ys ys' A (⊢ c₁ ⊔ ⊢ c₂)
proof (cases w ∈ set ws'')
  assume P: w ∈ set ws''
  show ?thesis
    apply (rule exI [of - f])
    apply (rule exI [of - s'])
    apply (rule conjI)
    apply (rule exI [of - t'])
    apply (rule ext)
  subgoal for x
  proof (cases [y←ys'. fst y = x], cases [y←ys. fst y = x] = [])
    case Cons
      thus ctying1-idem-lhs s t t' ys ys' x =
        ctying1-idem-rhs f s' t'' x
        by (insert fun-cong [OF I, of x], simp)
    next
    case Nil
      moreover case True
        ultimately show ctying1-idem-lhs s t t' ys ys' x =
          ctying1-idem-rhs f s' t'' x
          using H by (insert fun-cong [OF I, of x], simp)
    next
    case False
      hence [y←?x. fst y = x] ≠ [] ∨ [y←?xs. fst y = x] ≠ []
        using H by simp
      moreover {
        assume [y←?x. fst y = x] ≠ []
        hence [y←ys''. fst y = x] ≠ []
          using O and P by (auto simp: filter-concat)
        hence [y←?xs. fst y = x] ≠ []
          using J and K and Nil by simp
      }
      ultimately have Q: [y←?xs. fst y = x] ≠ [] ..
      hence (case snd (last [y←?xs. fst y = x]) of
        None ⇒ t x | Some i ⇒ i) = ctying1-idem-rhs f s' t'' x
        using Nil by (insert fun-cong [OF I, of x], simp)
      moreover have last [y←?xs. fst y = x] = last [y←ys. fst y = x]
        using H and Q by simp
      ultimately show ctying1-idem-lhs s t t' ys ys' x =
        ctying1-idem-rhs f s' t'' x
        using Nil and False by simp
  qed
  apply (rule conjI)
  subgoal
  proof -

```

```

show  $\forall x. (f\ x = [] \longleftrightarrow [y \leftarrow ys @ ys'.\ fst\ y = x] = []) \wedge$ 
 $(f\ x \neq [] \longrightarrow last\ (f\ x) = last\ [y \leftarrow ys @ ys'.\ fst\ y = x])$ 
(is  $\forall x. ?P\ x \wedge ?Q\ x)$ 
proof
  fix  $x$ 
  have  $?P\ x$ 
  proof
    assume  $Q: f\ x = []$ 
    hence  $[y \leftarrow ?xs @ ys'.\ fst\ y = x] = []$ 
    using  $J$  by simp
    moreover have  $[y \leftarrow ?x.\ fst\ y = x] = []$ 
    using  $K$  and  $O$  and  $P$  and  $Q$  by (simp add: filter-concat)
    ultimately show  $[y \leftarrow ys @ ys'.\ fst\ y = x] = []$ 
    using  $H$  by simp
  qed (insert H J, simp)
  moreover have  $?Q\ x$ 
  using  $J$  and  $H$  by simp
  ultimately show  $?P\ x \wedge ?Q\ x ..$ 
  qed
qed
by (insert K L M, blast)
next
assume  $P: w \notin set\ ws''$ 
let  $?y = fst\ (hd\ ws'')$ 
show ?thesis
proof (cases snd w = snd (hd ws''))
  assume  $Q: snd\ w = snd\ (hd\ ws'')$ 
  hence  $snd\ w \wedge snd\ (hd\ ws'') \vee \neg\ snd\ w \wedge \neg\ snd\ (hd\ ws'')$ 
  (is  $?P \vee ?Q)$ 
  by simp
  moreover {
    assume  $?P$ 
    have  $?x \in \vdash\ c_1$ 
    using  $G$  by (rule ctyping1-merge-aux.cases, insert <?P>, simp-all)
    moreover have  $?y \in \vdash\ c_1$ 
    using  $N$  by (rule ctyping1-merge-aux.cases, insert <?P>, simp-all)
    ultimately have ctyping1-idem-pred s t t' ?x ?y A ( $\vdash\ c_1$ )
    using  $A$  by simp
  }
  moreover {
    assume  $?Q$ 
    have  $?x \in \vdash\ c_2$ 
    using  $G$  by (rule ctyping1-merge-aux.cases, insert <?Q>, simp-all)
    moreover have  $?y \in \vdash\ c_2$ 
    using  $N$  by (rule ctyping1-merge-aux.cases, insert <?Q>, simp-all)
    ultimately have ctyping1-idem-pred s t t' ?x ?y A ( $\vdash\ c_2$ )
    using  $B$  by simp
  }
  }
ultimately obtain  $f_0$  and  $s_0$  and  $t_0$  and  $ys_0$  where

```

R: *ctyping1-idem-lhs* *s t t' ?x ?y* =
ctyping1-idem-rhs *f₀ s₀ t₀* **and**
S: $\forall x. (f_0 x = [] \leftrightarrow [y \leftarrow ?x @ ?y. fst y = x] = []) \wedge$
 $(f_0 x \neq [] \rightarrow last (f_0 x) = last [y \leftarrow ?x @ ?y. fst y = x])$ **and**
T: $f_0 = (\lambda x. [y \leftarrow ys_0. fst y = x])$ **and**
U: $ys_0 \in \vdash c_1 \wedge snd w \vee ys_0 \in \vdash c_2 \wedge \neg snd w$
by *auto*
from *U* **obtain** *w₀* **where**
V: $[w_0] \in \vdash c_1 \sqcup \vdash c_2$ **and**
W: $ys_0 = fst w_0$ **and**
X: $snd w_0 = snd w$
by *fastforce*
show *?thesis*
proof (*cases* $w_0 \in set ws''$)
assume *Y*: $w_0 \in set ws''$
show *?thesis*
apply (*rule* *exI* [*of* - *f*])
apply (*rule* *exI* [*of* - *s'*])
apply (*rule* *conjI*)
apply (*rule* *exI* [*of* - *t''*])
apply (*rule* *ext*)
subgoal for *x*
proof (*cases* $[y \leftarrow ys'. fst y = x]$, *cases* $[y \leftarrow ys. fst y = x] = []$)
case *Cons*
thus *ctyping1-idem-lhs* *s t t' ys ys' x* =
ctyping1-idem-rhs *f s' t'' x*
by (*insert fun-cong* [*OF I*, *of x*], *simp*)
next
case *Nil*
moreover case *True*
ultimately show *ctyping1-idem-lhs* *s t t' ys ys' x* =
ctyping1-idem-rhs *f s' t'' x*
using *H* **by** (*insert fun-cong* [*OF I*, *of x*], *simp*)
next
case *Nil*
case *False*
hence $[y \leftarrow ?x. fst y = x] \neq [] \vee [y \leftarrow ?xs. fst y = x] \neq []$
using *H* **by** *simp*
moreover {
assume $[y \leftarrow ?x. fst y = x] \neq []$
hence $[y \leftarrow ys_0. fst y = x] \neq []$
using *S* **and** *T* **by** *simp*
hence $[y \leftarrow ys''. fst y = x] \neq []$
using *O* **and** *W* **and** *Y* **by** (*auto simp: filter-concat*)
hence $[y \leftarrow ?xs. fst y = x] \neq []$
using *J* **and** *K* **and** *Nil* **by** *simp*
}
ultimately have *Z*: $[y \leftarrow ?xs. fst y = x] \neq []$..
hence (*case* *snd* (*last* $[y \leftarrow ?xs. fst y = x]$) *of*

```

    None  $\Rightarrow$   $t x \mid$  Some  $i \Rightarrow i$  = ctyping1-idem-rhs  $f s' t'' x$ 
    using Nil by (insert fun-cong [OF I, of x], simp)
  moreover have  $last [y \leftarrow ?xs. fst y = x] = last [y \leftarrow ys. fst y = x]$ 
    using H and Z by simp
  ultimately show ctyping1-idem-lhs  $s t t' ys ys' x =$ 
    ctyping1-idem-rhs  $f s' t'' x$ 
    using Nil and False by simp
qed
apply (rule conjI)
subgoal
proof -
  show  $\forall x. (f x = [] \longleftrightarrow [y \leftarrow ys @ ys'. fst y = x] = []) \wedge$ 
     $(f x \neq [] \longrightarrow last (f x) = last [y \leftarrow ys @ ys'. fst y = x])$ 
    (is  $\forall x. ?P x \wedge ?Q x$ )
  proof
    fix  $x$ 
    have  $?P x$ 
    proof
      assume  $Z: f x = []$ 
      hence  $[y \leftarrow ?xs @ ys'. fst y = x] = []$ 
        using J by simp
      moreover have  $[y \leftarrow ys''. fst y = x] = []$ 
        using K and Z by simp
      hence  $[y \leftarrow ys_0. fst y = x] = []$ 
        using O and W and Y by (simp add: filter-concat)
      hence  $[y \leftarrow ?x. fst y = x] = []$ 
        using S and T by simp
      ultimately show  $[y \leftarrow ys @ ys'. fst y = x] = []$ 
        using H by simp
    qed (insert H J, simp)
    moreover have  $?Q x$ 
      using J and H by simp
    ultimately show  $?P x \wedge ?Q x ..$ 
  qed
qed
by (insert K L M, blast)
next
assume  $Y: w_0 \notin set ws''$ 
let  $?ws = w_0 \# tl ws''$ 
{
  assume  $Z: tl ws'' \neq []$ 
  have  $tl ws'' \in \vdash c_1 \sqcup \vdash c_2$ 
    using N by (rule ctyping1-merge-aux.cases, insert Z, simp-all)
  moreover have  $snd (hd (tl ws'')) = (\neg snd w)$ 
    using N by (rule ctyping1-merge-aux.cases, insert Q Z, simp-all)
  moreover have  $w_0 \notin set (tl ws'')$ 
    using Y by (cases ws'', simp-all)
  ultimately have  $?ws \in \vdash c_1 \sqcup \vdash c_2$ 
    by (cases w_0, insert U W X, auto)
}

```

```

}
hence Z: ?ws ∈ ⊢ c1 ⊔ ⊢ c2
  by (cases tl ws'', insert V, simp-all)
let ?ys = concat (map fst (tl ws''))
let ?f = λx. [y←concat (map fst ?ws). fst y = x]
let ?t = λx. if f x = [] then t0 x else t'' x
have AA: ws'' = hd ws'' # tl ws''
  by (insert ctyping1-merge-aux-no-nil [OF N], simp)
have AB: ys'' = ?y @ ?ys
  using O by (subst (asm) AA, simp)
have AC: ∀x. [y←?ys. fst y = x] ≠ [] →
  last (?f x) = last (f x)
  using K and O by (subst (asm) AA, simp)
have AD: ∀x. [y←?ys. fst y = x] = [] ∧ [y←?y. fst y = x] ≠ [] →
  last (?f x) = last (f x)
  (is ∀x. ?P x ∧ ?Q x → -)
proof clarify
  fix x
  assume ?P x and ?Q x
  moreover from this and S and T have
    last [y←ys0. fst y = x] = last [y←?x @ ?y. fst y = x]
    by simp
  ultimately show last (?f x) = last (f x)
    using K and W and AB by simp
qed
show ?thesis
  apply (rule exI [of - ?f])
  apply (rule exI [of - s'])
  apply (rule conjI)
  apply (rule exI [of - ?t])
  apply (rule ext)
  subgoal for x
  proof (cases [y←ys'. fst y = x], cases [y←?xs. fst y = x] = [])
  case Cons
  hence AE:
    (case snd (last [y←ys'. fst y = x]) of
      None ⇒ t' x | Some i ⇒ i) =
    (case snd (last (f x)) of None ⇒ ?t x | Some i ⇒ i)
    using J by (insert fun-cong [OF I, of x], simp)
  show ctyping1-idem-lhs s t t' ys ys' x =
    ctyping1-idem-rhs ?f s' ?t x
  proof (cases [y←?ys. fst y = x] ≠ [])
  case True
  thus ?thesis
    using AC and AE and Cons by simp
  next
  case False
  moreover have [y←ys''. fst y = x] ≠ []
    using J and K and Cons by simp

```

ultimately have $[y \leftarrow ?y. \text{fst } y = x] \neq []$
 using *AB* by *simp*
 moreover from *this* have $?f x \neq []$
 using *S* and *T* and *W* by *simp*
 ultimately show *?thesis*
 using *AD* and *AE* and *Cons* and *False* by *simp*
 qed
 next
 case *Nil*
 moreover case *False*
 ultimately have
 (case *snd* (last $[y \leftarrow ys. \text{fst } y = x]$) of
 None $\Rightarrow t x \mid$ *Some* $i \Rightarrow i$) =
 (case *snd* (last $(f x)$) of *None* $\Rightarrow ?t x \mid$ *Some* $i \Rightarrow i$)
 using *J* and *H* by (insert *fun-cong* [*OF I*, of *x*], *simp*)
 moreover have
AE: $[y \leftarrow ?y. \text{fst } y = x] \neq [] \vee [y \leftarrow ?ys. \text{fst } y = x] \neq []$
 (is - $\vee ?P$)
 using *J* and *K* and *AB* and *False* by *auto*
 hence $?f x \neq []$
 using *S* and *T* and *W* by (cases *?P*, *simp-all*)
 moreover have *last* ($?f x$) = *last* ($f x$)
 using *AC* and *AD* and *AE* by *blast*
 ultimately show *ctyping1-idem-lhs* $s t t' ys ys' x =$
ctyping1-idem-rhs $?f s' ?t x$
 using *H* and *Nil* and *False* by *auto*
 next
 case *Nil*
 moreover case *True*
 ultimately have *AE*: $f x = []$
 using *J* by *simp*
 hence *AF*: $[y \leftarrow ?y @ ?ys. \text{fst } y = x] = []$
 using *K* and *AB* by *simp*
 show *ctyping1-idem-lhs* $s t t' ys ys' x =$
ctyping1-idem-rhs $?f s' ?t x$
 proof (cases $[y \leftarrow ?x. \text{fst } y = x] = []$)
 assume *AG*: $[y \leftarrow ?x. \text{fst } y = x] = []$
 moreover from *J* and *AE* have $s x = s' x$
 by (insert *fun-cong* [*OF I*, of *x*], *simp*)
 moreover have $[y \leftarrow ys_0. \text{fst } y = x] = []$
 using *S* and *T* and *AF* and *AG* by *simp*
 hence $?f x = []$
 using *W* and *AF* by *simp*
 ultimately show *?thesis*
 using *H* and *Nil* and *True* by *simp*
 next
 assume *AG*: $[y \leftarrow ?x. \text{fst } y = x] \neq []$
 moreover from *this* and *S* and *AE* and *AF* have
 (case *snd* (last $[y \leftarrow ?x. \text{fst } y = x]$) of

$None \Rightarrow t\ x \mid Some\ i \Rightarrow i) =$
 $(case\ snd\ (last\ (f_0\ x))\ of\ None \Rightarrow ?t\ x \mid Some\ i \Rightarrow i)$
by $(insert\ fun-cong\ [OF\ R,\ of\ x],\ simp)$
moreover have $[y \leftarrow ys_0.\ fst\ y = x] \neq []$
using S and T and AG **by** $simp$
hence $?f\ x \neq []$
using W **by** $simp$
moreover have $last\ (?f\ x) = last\ (f_0\ x)$
using T and W and AF **by** $simp$
ultimately show $?thesis$
using H and Nil and $True$ **by** $auto$
qed
qed
apply $(rule\ conjI)$
subgoal
proof $-$
show $\forall x.\ (?f\ x = [] \longleftrightarrow [y \leftarrow ys \ @\ ys'.\ fst\ y = x] = []) \wedge$
 $(?f\ x \neq [] \longrightarrow last\ (?f\ x) = last\ [y \leftarrow ys \ @\ ys'.\ fst\ y = x])$
(is $\forall x.\ ?P\ x \wedge ?Q\ x)$
proof
fix x
have $AE: ?P\ x$
proof
assume $AF: ?f\ x = []$
hence $[y \leftarrow ?x \ @\ ?y.\ fst\ y = x] = []$
using S and T and W **by** $simp$
moreover from $this$ and J and K and AB and AF **have**
 $[y \leftarrow ?xs \ @\ ys'.\ fst\ y = x] = []$
by $auto$
ultimately show $[y \leftarrow ys \ @\ ys'.\ fst\ y = x] = []$
using H **by** $simp$
next
assume $[y \leftarrow ys \ @\ ys'.\ fst\ y = x] = []$
hence $[y \leftarrow ?x \ @\ ?y \ @\ ?ys.\ fst\ y = x] = []$
using H and J and K and AB **by** $simp$
moreover from $this$ **have** $[y \leftarrow ys_0.\ fst\ y = x] = []$
using S and T **by** $simp$
ultimately show $?f\ x = []$
using W **by** $simp$
qed
moreover have $?Q\ x$
proof $(clarify,\ cases\ [y \leftarrow ?y \ @\ ?ys.\ fst\ y = x])$
case Nil
hence $last\ (?f\ x) = last\ (f_0\ x)$
using T and W **by** $simp$
moreover assume $?f\ x \neq []$
hence $[y \leftarrow ys \ @\ ys'.\ fst\ y = x] \neq []$
using AE **by** $blast$
hence $[y \leftarrow ?x \ @\ ?y \ @\ ?ys.\ fst\ y = x] \neq []$

```

    using H and J and K and AB by simp
  ultimately have  $\text{last } (?f\ x) = \text{last } [y \leftarrow ?x. \text{fst } y = x]$ 
    using S and Nil by simp
  moreover have  $[y \leftarrow ?xs @ ys'. \text{fst } y = x] = []$ 
    using J and K and AB and Nil by simp
  ultimately show
     $\text{last } (?f\ x) = \text{last } [y \leftarrow ys @ ys'. \text{fst } y = x]$ 
    using H by simp
next
  case Cons
  hence  $[y \leftarrow ?y. \text{fst } y = x] \neq [] \vee$ 
     $[y \leftarrow ?ys. \text{fst } y = x] \neq []$ 
    by auto
  hence  $\text{last } (?f\ x) = \text{last } (f\ x)$ 
    using AC and AD by blast
  moreover have  $f\ x \neq []$ 
    using K and AB and Cons by simp
  ultimately show
     $\text{last } (?f\ x) = \text{last } [y \leftarrow ys @ ys'. \text{fst } y = x]$ 
    using H and J by simp
  qed
  ultimately show  $?P\ x \wedge ?Q\ x ..$ 
  qed
  qed
  by (rule conjI, rule exI [of - concat (map fst ?ws)],
    insert M Z, auto simp only: ctyping1-merge-def)
  qed
next
  assume  $\text{snd } w \neq \text{snd } (\text{hd } ws'')$ 
  hence  $\text{snd } w \wedge \neg \text{snd } (\text{hd } ws'') \vee \neg \text{snd } w \wedge \text{snd } (\text{hd } ws'')$ 
    (is ?P  $\vee$  ?Q)
    by simp
  moreover {
    assume ?P
    moreover have  $?x \in \vdash c_1$ 
      using G by (rule ctyping1-merge-aux.cases, insert  $\langle ?P \rangle$ , simp-all)
    moreover have  $(?x, \text{True}) \notin \text{set } ws''$ 
      using P and  $\langle ?P \rangle$  by (cases w, simp)
    ultimately have  $w \# ws'' \in \vdash c_1 \sqcup \vdash c_2$ 
      using N by (cases w, auto)
  }
  moreover {
    assume ?Q
    moreover have  $?x \in \vdash c_2$ 
      using G by (rule ctyping1-merge-aux.cases, insert  $\langle ?Q \rangle$ , simp-all)
    moreover have  $(?x, \text{False}) \notin \text{set } ws''$ 
      using P and  $\langle ?Q \rangle$  by (cases w, simp)
    ultimately have  $w \# ws'' \in \vdash c_1 \sqcup \vdash c_2$ 
      using N by (cases w, auto)
  }

```

```

}
ultimately have Q: w # ws'' ∈ ⊢ c₁ [] ⊢ c₂
  (is ?ws ∈ -) ..
let ?f = λx. [y←concat (map fst ?ws). fst y = x]
let ?t = λx. if f x = [] then t x else t' x
show ?thesis
  apply (rule exI [of - ?f])
  apply (rule exI [of - s'])
  apply (rule conjI)
  apply (rule exI [of - ?t])
  apply (rule ext)
  subgoal for x
  proof (cases [y←ys'. fst y = x], cases [y←?xs. fst y = x] = [])
    case Cons
      moreover from this have
        (case snd (last [y←ys'. fst y = x]) of
          None ⇒ t' x | Some i ⇒ i) =
          (case snd (last (f x)) of None ⇒ ?t x | Some i ⇒ i)
        using J by (insert fun-cong [OF I, of x], simp)
      moreover have ?f x ≠ []
        using J and K and O and Cons by simp
      moreover have f x ≠ []
        using J and Cons by simp
      hence last (?f x) = last (f x)
        using K and O by simp
      ultimately show ctying1-idem-lhs s t t' ys ys' x =
        ctying1-idem-rhs ?f s' ?t x
        by auto
    next
    case Nil
      moreover case False
      ultimately have
        (case snd (last [y←ys. fst y = x]) of
          None ⇒ t x | Some i ⇒ i) =
          (case snd (last (f x)) of None ⇒ ?t x | Some i ⇒ i)
        using J and H by (insert fun-cong [OF I, of x], simp)
      moreover have ?f x ≠ []
        using J and K and O and False by simp
      moreover have f x ≠ []
        using J and False by simp
      hence last (?f x) = last (f x)
        using K and O by simp
      ultimately show ctying1-idem-lhs s t t' ys ys' x =
        ctying1-idem-rhs ?f s' ?t x
        using H and Nil and False by auto
  next
  case Nil
    moreover case True
    ultimately have R: f x = []

```

using J **by** *simp*
show *ctyping1-idem-lhs* $s\ t\ t'\ ys\ ys'\ x =$
ctyping1-idem-rhs $?f\ s'\ ?t\ x$
proof (*cases* $[y \leftarrow ?x.\ fst\ y = x] = []$)
assume $[y \leftarrow ?x.\ fst\ y = x] = []$
moreover **have** $[y \leftarrow ys''.\ fst\ y = x] = []$
using K **and** R **by** *simp*
ultimately **have** $?f\ x = []$
using O **by** *simp*
moreover **from** J **and** R **have** $s\ x = s'\ x$
by (*insert fun-cong* $[OF\ I,\ of\ x],\ simp$)
ultimately **show** *?thesis*
using H **and** Nil **and** $True$ **by** *simp*
next
assume $[y \leftarrow ?x.\ fst\ y = x] \neq []$
moreover **have** $last\ [y \leftarrow ys.\ fst\ y = x] = last\ [y \leftarrow ?x.\ fst\ y = x]$
using H **and** $True$ **by** *simp*
moreover **have** $last\ (?f\ x) = last\ [y \leftarrow ?x.\ fst\ y = x]$
using K **and** O **and** R **by** *simp*
ultimately **show** *?thesis*
using H **and** R **and** Nil **by** *simp*
qed
qed
apply (*rule conjI*)
subgoal
proof –
show $\forall x.\ (?f\ x = [] \longleftrightarrow [y \leftarrow ys\ @\ ys'.\ fst\ y = x] = []) \wedge$
 $(?f\ x \neq [] \longrightarrow last\ (?f\ x) = last\ [y \leftarrow ys\ @\ ys'.\ fst\ y = x])$
(is $\forall x.\ ?P\ x \wedge ?Q\ x$)
proof
fix x
have $?P\ x$
proof
assume $?f\ x = []$
hence $[y \leftarrow ?x\ @\ ys''.\ fst\ y = x] = []$
using O **by** *simp*
moreover **from** *this* **have** $[y \leftarrow ?xs\ @\ ys'.\ fst\ y = x] = []$
using J **and** K **by** *simp*
ultimately **show** $[y \leftarrow ys\ @\ ys'.\ fst\ y = x] = []$
using H **by** *simp*
next
assume $[y \leftarrow ys\ @\ ys'.\ fst\ y = x] = []$
hence $[y \leftarrow ?x\ @\ ?xs\ @\ ys'.\ fst\ y = x] = []$
using H **by** *simp*
moreover **from** *this* **have** $[y \leftarrow ys''.\ fst\ y = x] = []$
using J **and** K **by** *simp*
ultimately **show** $?f\ x = []$
using O **by** *simp*
qed

moreover have $?Q\ x$
proof (*clarify, cases* $[y \leftarrow ys''.\ \text{fst}\ y = x]$)
case *Nil*
hence $\text{last}\ (?f\ x) = \text{last}\ [y \leftarrow ?x.\ \text{fst}\ y = x]$
using *O* **by** *simp*
moreover have $[y \leftarrow ?xs\ @\ ys'.\ \text{fst}\ y = x] = []$
using *J* **and** *K* **and** *Nil* **by** *simp*
hence
 $\text{last}\ [y \leftarrow ys\ @\ ys'.\ \text{fst}\ y = x] = \text{last}\ [y \leftarrow ?x.\ \text{fst}\ y = x]$
using *H* **by** *simp*
ultimately show
 $\text{last}\ (?f\ x) = \text{last}\ [y \leftarrow ys\ @\ ys'.\ \text{fst}\ y = x]$
by *simp*
next
case *Cons*
hence $\text{last}\ (?f\ x) = \text{last}\ (f\ x)$
using *K* **and** *O* **by** *simp*
moreover have *R*: $f\ x \neq []$
using *K* **and** *Cons* **by** *simp*
hence $\text{last}\ [y \leftarrow ?xs\ @\ ys'.\ \text{fst}\ y = x] = \text{last}\ (f\ x)$
using *J* **by** *simp*
moreover have $[y \leftarrow ?xs\ @\ ys'.\ \text{fst}\ y = x] \neq []$
using *J* **and** *R* **by** *simp*
ultimately show
 $\text{last}\ (?f\ x) = \text{last}\ [y \leftarrow ys\ @\ ys'.\ \text{fst}\ y = x]$
using *H* **by** *simp*
qed
ultimately show $?P\ x \wedge ?Q\ x ..$
qed
qed
by (*rule* *conjI*, *rule* *exI* [*of* - *concat* (*map* *fst* *?ws*)],
insert *M* *Q*, *auto* *simp* *only*: *ctyping1-merge-def*)
qed
qed
qed
qed

lemma *ctyping1-merge-append-idem-fst*:
assumes
A: $\bigwedge ys\ ys'.\ ys \in \vdash c_1 \implies ys' \in \vdash c_1 \implies$
 $\text{ctyping1-idem-pred}\ s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_1)$ **and**
B: $\bigwedge ys\ ys'.\ ys \in \vdash c_2 \implies ys' \in \vdash c_2 \implies$
 $\text{ctyping1-idem-pred}\ s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_2)$ **and**
C: $s \in A$ **and**
D: $ys \in \vdash c_1 \sqcup_{@} \vdash c_2$ **and**
E: $ys' \in \vdash c_1 \sqcup_{@} \vdash c_2$
shows $\text{ctyping1-idem-pred}\ s\ t\ t'\ ys\ ys'\ A\ (\vdash\ c_1 \sqcup_{@} \vdash c_2)$
apply (*subst* *ctyping1-merge-append-def*)
apply (*split* *if-split*)

apply (*rule conjI*)
subgoal
proof
 assume $F: \text{card } (\vdash c_2) = \text{Suc } 0$
 with D obtain ys_1 and ys_2 where
 $G: ys = ys_1 @ ys_2$ and
 $H: ys_1 \in \vdash c_1$ and
 $I: ys_2 \in \vdash c_2$
 by (*auto simp: ctying1-merge-append-def ctying1-append-def*)
 from E and F obtain ys_1' and ys_2' where
 $J: ys' = ys_1' @ ys_2'$ and
 $K: ys_1' \in \vdash c_1$ and
 $L: ys_2' \in \vdash c_2$
 by (*auto simp: ctying1-merge-append-def ctying1-append-def*)
 have $M: ys_2' = ys_2$
 using F and I and L by (*fastforce simp: card-1-singleton-iff*)
 obtain f and s' and t'' and ys_1'' where
 $N: \text{ctying1-idem-lhs } s \ t \ t' \ ys_1 \ ys_1' =$
 $\text{ctying1-idem-rhs } f \ s' \ t''$ and
 $O: \forall x. (f \ x = [] \longleftrightarrow [y \leftarrow ys_1 @ ys_1'. \text{fst } y = x] = []) \wedge$
 $(f \ x \neq [] \longrightarrow \text{last } (f \ x) = \text{last } [y \leftarrow ys_1 @ ys_1'. \text{fst } y = x])$ and
 $P: f = (\lambda x. [y \leftarrow ys_1''. \text{fst } y = x])$ and
 $Q: ys_1'' \in \vdash c_1$ and
 $R: s' \in A$
 using A [$OF \ H \ K$] by *auto*
 let $?f = \lambda x. [y \leftarrow ys_1'' @ ys_2. \text{fst } y = x]$
 let $?t = \lambda x. \text{if } [y \leftarrow ys_2. \text{fst } y = x] = [] \text{ then } t'' \ x \ \text{else } t' \ x$
 show *ctying1-idem-pred* $s \ t \ t' \ ys \ ys' \ A \ (\vdash c_1 @ \vdash c_2)$
 apply (*rule exI [of - ?f]*)
 apply (*rule exI [of - s']*)
 apply (*rule conjI*)
 apply (*rule exI [of - ?t]*)
 apply (*rule ext*)
 subgoal for x
 proof (*cases* $[y \leftarrow ys_2. \text{fst } y = x]$, *cases* $f \ x = []$)
 case *Nil*
 moreover case *True*
 ultimately have $s \ x = s' \ x$
 using O by (*insert fun-cong [OF N, of x], simp*)
 moreover have $[y \leftarrow ys'. \text{fst } y = x] = []$
 using J and M and O and *Nil* and *True* by *simp*
 moreover have $[y \leftarrow ys. \text{fst } y = x] = []$
 using G and O and *Nil* and *True* by *simp*
 moreover have $?f \ x = []$
 using P and *Nil* and *True* by *simp*
 ultimately show *ctying1-idem-lhs* $s \ t \ t' \ ys \ ys' \ x =$
 ctying1-idem-rhs $?f \ s' \ ?t \ x$
 by *simp*
 next

case Nil
moreover from this have
 $[y \leftarrow ys'. \text{fst } y = x] = [y \leftarrow ys_1'. \text{fst } y = x]$
using J and M by simp
moreover have $[y \leftarrow ys. \text{fst } y = x] = [y \leftarrow ys_1. \text{fst } y = x]$
using G and Nil by simp
moreover case False
moreover from this have $?f \ x \neq []$
using P by simp
moreover have $\text{last } (?f \ x) = \text{last } (f \ x)$
using P and Nil by simp
ultimately show *ctyping1-idem-lhs* $s \ t \ t' \ ys \ ys' \ x =$
ctyping1-idem-rhs $?f \ s' \ ?t \ x$
by (*insert fun-cong [OF N, of x], auto*)

next
case Cons
moreover from this have $[y \leftarrow ys'. \text{fst } y = x] \neq []$
using J and M by simp
moreover have
 $\text{last } [y \leftarrow ys'. \text{fst } y = x] = \text{last } [y \leftarrow ys_2. \text{fst } y = x]$
using J and M and Cons by simp
ultimately show *ctyping1-idem-lhs* $s \ t \ t' \ ys \ ys' \ x =$
ctyping1-idem-rhs $?f \ s' \ ?t \ x$
by simp

qed
apply (*rule conjI*)
subgoal
proof –
show $\forall x. (?f \ x = [] \longleftrightarrow [y \leftarrow ys \ @ \ ys'. \text{fst } y = x] = []) \wedge$
 $(?f \ x \neq [] \longrightarrow \text{last } (?f \ x) = \text{last } [y \leftarrow ys \ @ \ ys'. \text{fst } y = x])$
(is $\forall x. ?P \ x \wedge ?Q \ x$)

proof
fix x
have $?P \ x$
using G and J and M and O and P by auto
moreover have $?Q \ x$
proof (*clarify, cases [y ← ys₂. fst y = x]*)
case Nil
moreover assume $?f \ x \neq []$
ultimately have
 $\text{last } (?f \ x) = \text{last } [y \leftarrow ys_1 \ @ \ ys_1'. \text{fst } y = x]$
using O and P by simp
thus $\text{last } (?f \ x) = \text{last } [y \leftarrow ys \ @ \ ys'. \text{fst } y = x]$
using G and J and M and Nil by simp

next
case Cons
thus $\text{last } (?f \ x) = \text{last } [y \leftarrow ys \ @ \ ys'. \text{fst } y = x]$
using J and M by simp

qed

ultimately show $?P\ x \wedge ?Q\ x \dots$
 qed
 qed
 by (rule conjI, rule exI [of - $ys_1'' @ ys_2$],
 insert I Q R, auto simp: ctyping1-append-def)
 qed
 subgoal
 proof
 assume $F: \text{card } (\vdash c_2) \neq \text{Suc } 0$
 with D obtain ws and xs where
 G: $ys = ws @ xs$ and
 H: $ws \in \vdash c_1 \sqcup \vdash c_2$ and
 I: $xs \in \vdash c_2$
 by (auto simp: ctyping1-merge-append-def ctyping1-append-def)
 from E and F obtain ws' and xs' where
 J: $ys' = ws' @ xs'$ and
 K: $ws' \in \vdash c_1 \sqcup \vdash c_2$ and
 L: $xs' \in \vdash c_2$
 by (auto simp: ctyping1-merge-append-def ctyping1-append-def)
 from I have $[(xs, \text{False})] \in \vdash c_1 \sqcup \vdash c_2 \dots$
 hence $M: xs \in \vdash c_1 \sqcup \vdash c_2$
 by (force simp: ctyping1-merge-def)
 obtain f and s' and r and zs where
 N: ctyping1-idem-lhs s t t' ws xs =
 ctyping1-idem-rhs f s' r and
 O: $\forall x. (f\ x = [] \longleftrightarrow [y \leftarrow ws @ xs. \text{fst } y = x] = []) \wedge$
 $(f\ x \neq [] \longrightarrow \text{last } (f\ x) = \text{last } [y \leftarrow ws @ xs. \text{fst } y = x])$ and
 P: $f = (\lambda x. [y \leftarrow zs. \text{fst } y = x])$ and
 Q: $zs \in \vdash c_1 \sqcup \vdash c_2$ and
 R: $s' \in A$
 using ctyping1-merge-idem-fst [OF A B C H M] by auto
 obtain f' and s'' and r' and zs' where
 S: ctyping1-idem-lhs s t t' zs ws' =
 ctyping1-idem-rhs f' s'' r' and
 T: $\forall x. (f'\ x = [] \longleftrightarrow [y \leftarrow zs @ ws'. \text{fst } y = x] = []) \wedge$
 $(f'\ x \neq [] \longrightarrow \text{last } (f'\ x) = \text{last } [y \leftarrow zs @ ws'. \text{fst } y = x])$ and
 U: $f' = (\lambda x. [y \leftarrow zs'. \text{fst } y = x])$ and
 V: $zs' \in \vdash c_1 \sqcup \vdash c_2$ and
 W: $s'' \in A$
 using ctyping1-merge-idem-fst [OF A B C Q K] by auto
 let $?f = \lambda x. [y \leftarrow zs' @ xs'. \text{fst } y = x]$
 let $?t = \lambda x. \text{if } [y \leftarrow xs'. \text{fst } y = x] = [] \text{ then } r'\ x \text{ else } t'\ x$
 show ctyping1-idem-pred s t t' ys ys' A ($\vdash c_1 \sqcup \vdash c_2 @ \vdash c_2$)
 apply (rule exI [of - ?f])
 apply (rule exI [of - s''])
 apply (rule conjI)
 apply (rule exI [of - ?t])
 apply (rule ext)
 subgoal for x

proof (*cases* $[y \leftarrow xs'. \text{fst } y = x]$, *cases* $f' x = []$)
case *Nil*
moreover case *True*
hence $s x = s'' x$
using *T* **by** (*insert fun-cong* [*OF S*, *of x*], *simp*)
moreover have $[y \leftarrow ys'. \text{fst } y = x] = []$
using *J* **and** *T* **and** *Nil* **and** *True* **by** *simp*
moreover have $[y \leftarrow zs. \text{fst } y = x] = []$
using *T* **and** *True* **by** *simp*
hence $[y \leftarrow ys. \text{fst } y = x] = []$
using *G* **and** *O* **and** *P* **by** *simp*
moreover have $?f x = []$
using *U* **and** *Nil* **and** *True* **by** *simp*
ultimately show *ctyping1-idem-lhs* $s t t' ys ys' x =$
ctyping1-idem-rhs $?f s'' ?t x$
by *simp*

next
case *Nil*
moreover from *this* **have**
 $X: [y \leftarrow ys'. \text{fst } y = x] = [y \leftarrow ws'. \text{fst } y = x]$
using *J* **by** *simp*
moreover case *False*
moreover have
 $[y \leftarrow zs. \text{fst } y = x] \neq [] \wedge [y \leftarrow ys. \text{fst } y = x] \neq [] \wedge$
 $\text{last } [y \leftarrow ys. \text{fst } y = x] = \text{last } [y \leftarrow zs. \text{fst } y = x]$
(is $?P \wedge ?Q \wedge ?R$) **if**
 $a: [y \leftarrow ys'. \text{fst } y = x] = []$

proof –
have $?P$
using *T* **and** *X* **and** *False* **and** *a* **by** *simp*
moreover from *this* **have** $?Q$
using *G* **and** *O* **and** *P* **by** *simp*
moreover have $?R$
using *G* **and** *O* **and** *P* **and** $\langle ?P \rangle$ **by** *simp*
ultimately show *?thesis*
by *simp*

qed
moreover have $?f x \neq []$
using *U* **and** *False* **by** *simp*
moreover have $\text{last } (?f x) = \text{last } (f' x)$
using *U* **and** *Nil* **by** *simp*
ultimately show *ctyping1-idem-lhs* $s t t' ys ys' x =$
ctyping1-idem-rhs $?f s'' ?t x$
by (*insert fun-cong* [*OF S*, *of x*], *auto*)

next
case *Cons*
moreover from *this* **have** $[y \leftarrow ys'. \text{fst } y = x] \neq []$
using *J* **by** *simp*
moreover have

$last [y \leftarrow ys'. fst y = x] = last [y \leftarrow xs'. fst y = x]$
using J and $Cons$ **by** $simp$
ultimately show $ctyping1-idem-lhs s t t' ys ys' x =$
 $ctyping1-idem-rhs ?f s'' ?t x$
by $simp$
qed
apply ($rule conjI$)
subgoal
proof –
show $\forall x. (?f x = [] \longleftrightarrow [y \leftarrow ys @ ys'. fst y = x] = []) \wedge$
 $(?f x \neq [] \longrightarrow last (?f x) = last [y \leftarrow ys @ ys'. fst y = x])$
(is $\forall x. ?P x \wedge ?Q x)$
proof
fix x
have $?P x$
proof
assume $[y \leftarrow zs' @ xs'. fst y = x] = []$
moreover from this have $[y \leftarrow zs @ ws'. fst y = x] = []$
using T and U **by** $simp$
moreover from this have $[y \leftarrow ws @ xs. fst y = x] = []$
using O and P **by** $simp$
ultimately show $[y \leftarrow ys @ ys'. fst y = x] = []$
using G and J **by** $simp$
next
assume $[y \leftarrow ys @ ys'. fst y = x] = []$
hence $[y \leftarrow ws @ xs @ ws' @ xs'. fst y = x] = []$
using G and J **by** $simp$
moreover from this have $[y \leftarrow zs. fst y = x] = []$
using O and P **by** $simp$
ultimately show $[y \leftarrow zs' @ xs'. fst y = x] = []$
using T and U **by** $simp$
qed
moreover have $?Q x$
proof ($clarify$, $cases [y \leftarrow xs'. fst y = x]$)
case Nil
moreover assume $?f x \neq []$
ultimately have $X: f' x \neq []$
using U **by** $simp$
hence $Y: last (?f x) = last [y \leftarrow zs @ ws'. fst y = x]$
using T and U and Nil **by** $simp$
show $last (?f x) = last [y \leftarrow ys @ ys'. fst y = x]$
proof ($cases [y \leftarrow ws'. fst y = x] = []$)
case True
moreover from this have $f x \neq []$
using P and T and X **by** $simp$
ultimately have
 $last (?f x) = last [y \leftarrow ws @ xs. fst y = x]$
using O and P and Y **by** $simp$
thus $?thesis$

```

    using G and J and Nil and True by simp
  next
    case False
    thus ?thesis
      using J and Y and Nil by simp
    qed
  qed (simp add: J)
  ultimately show ?P x ∧ ?Q x ..
  qed
  qed
  by (rule conjI, rule exI [of - zs' @ xs'],
      insert L V W, auto simp: ctyping1-append-def)
  qed
done

```

lemma *ctyping1-aux-idem-fst*:

```

[[s ∈ A; ys ∈ ⊢ c; ys' ∈ ⊢ c]] ⇒
  ctyping1-idem-pred s t t' ys ys' A (⊢ c)
proof (induction c arbitrary: ys ys')
  fix c1 c2 ys ys'
  show
    [[∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c1 ⇒ ys' ∈ ⊢ c1 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c1);
      ∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c2 ⇒ ys' ∈ ⊢ c2 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c2);
      s ∈ A; ys ∈ ⊢ c1; c2; ys' ∈ ⊢ c1; c2]] ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c1; c2)
    by (simp, rule ctyping1-merge-append-idem-fst [simplified])
  next
  fix c1 c2 ys ys'
  show
    [[∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c1 ⇒ ys' ∈ ⊢ c1 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c1);
      ∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c2 ⇒ ys' ∈ ⊢ c2 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c2);
      s ∈ A; ys ∈ ⊢ c1 OR c2; ys' ∈ ⊢ c1 OR c2]] ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c1 OR c2)
    by (simp, rule ctyping1-merge-idem-fst [simplified])
  next
  fix b c1 c2 ys ys'
  assume
    A: ∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c1 ⇒ ys' ∈ ⊢ c1 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c1) and
    B: ∧ ys ys'. s ∈ A ⇒ ys ∈ ⊢ c2 ⇒ ys' ∈ ⊢ c2 ⇒
      ctyping1-idem-pred s t t' ys ys' A (⊢ c2) and
    C: s ∈ A and
    D: ys ∈ ⊢ IF b THEN c1 ELSE c2 and
    E: ys' ∈ ⊢ IF b THEN c1 ELSE c2

```

```

show ctyping1-idem-pred s t t' ys ys' A ( $\vdash$  IF b THEN c1 ELSE c2)
proof (cases  $\vdash b$ )
  case None
  show ?thesis
  by (insert A B C D E None, simp,
    rule ctyping1-merge-idem-fst [simplified])
next
  case (Some v)
  show ?thesis
  proof (cases v)
  case True
  thus ?thesis
  by (insert A C D E Some, simp add: ctyping1-merge-empty-rhs)
next
  case False
  thus ?thesis
  by (insert B C D E Some, simp add: ctyping1-merge-empty-lhs)
qed
qed
next
fix b c ys ys'
assume
  A:  $\bigwedge ys\ ys'. s \in A \implies ys \in \vdash c \implies ys' \in \vdash c \implies$ 
    ctyping1-idem-pred s t t' ys ys' A ( $\vdash c$ ) and
  B:  $s \in A$  and
  C:  $ys \in \vdash \text{WHILE } b \text{ DO } c$  and
  D:  $ys' \in \vdash \text{WHILE } b \text{ DO } c$ 
have E: ctyping1-idem-pred s t t' ys ys' A ( $\vdash \text{WHILE } b \text{ DO } c$ ) if
  a:  $ys \in \vdash c$  and
  b:  $ys' \in \vdash c$  and
  c:  $\vdash b \in \{\text{Some True, None}\}$ 
proof –
  have ctyping1-idem-pred s t t' ys ys' A ( $\vdash c$ )
  using A and B and a and b by simp
  then obtain f and s' and t'' and ys'' where
  E: ctyping1-idem-lhs s t t' ys ys' =
    ctyping1-idem-rhs f s' t'' and
  F:  $\forall x. (f\ x = [] \iff [y \leftarrow ys @ ys'. \text{fst } y = x] = []) \wedge$ 
    ( $f\ x \neq [] \implies \text{last } (f\ x) = \text{last } [y \leftarrow ys @ ys'. \text{fst } y = x]$ ) and
  G:  $f = (\lambda x. [y \leftarrow ys''. \text{fst } y = x])$  and
  H:  $ys'' \in \vdash c$  and
  I:  $s' \in A$ 
  by auto
  show ?thesis
  by (rule exI [of - f], insert E F G H I c, force)
qed
show ctyping1-idem-pred s t t' ys ys' A ( $\vdash \text{WHILE } b \text{ DO } c$ )
proof (cases  $\vdash b$ )
  case None

```

```

show ?thesis
proof (cases ys')
  case Nil
  show ?thesis
  proof (cases ys = [])
    case True
    thus ?thesis
    by (insert B None Nil, force)
  next
  case False
  thus ?thesis
  by (insert B C None Nil, force)
qed
next
case Cons
show ?thesis
proof (cases ys = [])
  case True
  show ?thesis
  apply (insert B D None Cons True)
  apply (rule exI [of - λx. [y←ys'. fst y = x]])
  apply (rule exI [of - s])
  apply (rule conjI)
  apply fastforce
  apply (rule conjI)
  apply fastforce
  apply (rule conjI)
  apply (rule exI [of - ys'])
  by simp-all
  next
  case False
  hence ys ∈ ⊢ c ∧ ys' ∈ ⊢ c
  using C and D and None and Cons by simp
  thus ?thesis
  using None by (blast intro: E)
qed
qed
next
case (Some v)
show ?thesis
proof (cases v)
  case True
  hence ys ∈ ⊢ c ∧ ys' ∈ ⊢ c
  using C and D and Some by simp
  thus ?thesis
  using Some and True by (fastforce intro: E)
next
case False
hence ys = [] ∧ ys' = []

```

using *C* and *D* and *Some* **by** *simp*
thus *?thesis*
by (*insert B Some False, simp*)
qed
qed
qed *fastforce+*

lemma *ctyping1-idem-fst-1*:

$\llbracket s \in A; ys \in \vdash c; ys' \in \vdash c \rrbracket \implies \exists f s'$
 $(\exists t''. \text{ctyping1-idem-lhs } s \ t \ t' \ ys \ ys' = \text{ctyping1-idem-rhs } f \ s' \ t'') \wedge$
 $(\exists ys''. f = (\lambda x. [y \leftarrow ys''. \text{fst } y = x]) \wedge ys'' \in \vdash c) \wedge s' \in A$
apply (*drule ctyping1-aux-idem-fst [where ys' = ys'], assumption+*)
apply *clarify*
apply (*rule exI, (rule conjI)?+*)
apply *assumption*
by *blast*

lemma *ctyping1-idem-fst-2*:

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

lemma *ctyping1-idem-fst*:

$\vdash c (\subseteq A, X) = (B, Y) \implies \text{case } \vdash c (\subseteq B, Y) \text{ of } (B', Y') \Rightarrow B' = B$
by (*auto intro: ctyping1-idem-fst-1 ctyping1-idem-fst-2 simp: ctyping1-def*)

lemma *ctyping1-idem-snd-1*:

assumes

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

B: $\forall r f s.$

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

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

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

shows $UNIV = S$
proof –
obtain s **where** $C: s \in A$
using A **by** *blast*
obtain ys **where** $D: ys \in \vdash c$
by (*insert ctying1-aux-nonempty, blast*)
let $?f = \lambda x. [y \leftarrow ys. fst\ y = x]$
show *?thesis*
using B [*rule-format, of ?r ?f s* ($\lambda x. 0$) *?f s*] **and** C **and** D **by** *auto*
qed

lemma *ctying1-idem-snd-2*:

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

by (*rule equalityI, force+*)

lemma *ctying1-idem-snd*:

$\vdash c (\subseteq A, X) = (B, Y) \implies case\ \vdash c (\subseteq B, Y)$ *of* $(B', Y') \Rightarrow Y' = Y$
by (*clarsimp simp: ctying1-def ctying1-idem-snd-1 ctying1-idem-snd-2*)

lemma *ctying1-idem*:

$\vdash c (\subseteq A, X) = (B, Y) \implies \vdash c (\subseteq B, Y) = (B, Y)$
by (*frule ctying1-idem-fst, drule ctying1-idem-snd, auto*)

end

end

4 Overapproximation of program semantics by the main type system

theory *Overapproximation*

imports *Idempotence*

begin

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

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

4.1 Global context proofs

lemma *avars-aval*:

$s = t (\subseteq \text{avars } a) \Longrightarrow \text{aval } a \ s = \text{aval } a \ t$
by (*induction a, simp-all*)

4.2 Local context proofs

context *noninterf*
begin

lemma *interf-set-mono*:

$\llbracket A' \subseteq A; X \subseteq X'; \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y; \forall (B, Y) \in \text{insert } (\text{Univ? } A \ X, Z) \ U. B: Y \rightsquigarrow W \rrbracket \Longrightarrow$
 $\forall (B, Y) \in \text{insert } (\text{Univ? } A' \ X', Z) \ U'. B: Y \rightsquigarrow W$
by (*subgoal-tac Univ? A' X' \subseteq Univ? A X, fastforce, auto simp: univ-states-if-def*)

lemma *btying1-btying2-aux-1* [*elim*]:

assumes

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

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

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

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

proof –

have $\text{aval } a_1 \ s = \text{aval } a_1 (\lambda x. 0) \wedge \text{aval } a_2 \ s = \text{aval } a_2 (\lambda x. 0)$

using A **and** B **by** (*blast intro: avars-aval*)

thus *?thesis*

using C **by** *simp*

qed

lemma *btying1-btying2-aux-2* [*elim*]:

assumes

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

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

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

$D: \text{aval } a_1 \ s < \text{aval } a_2 \ s$
shows *False*
proof –
have $\text{aval } a_1 \ s = \text{aval } a_1 \ (\lambda x. 0) \wedge \text{aval } a_2 \ s = \text{aval } a_2 \ (\lambda x. 0)$
using *A and B by (blast intro: avars-aval)*
thus *?thesis*
using *C and D by simp*
qed

lemma *btyping1-btyping2-aux:*
 $\vdash b = \text{Some } v \implies \Vdash b (\subseteq A, X) = \text{Some } (\text{if } v \text{ then } A \text{ else } \{\})$
by (*induction b arbitrary: v, auto split: if-split-asm option.split-asm*)

lemma *btyping1-btyping2:*
 $\vdash b = \text{Some } v \implies \Vdash b (\subseteq A, X) = (\text{if } v \text{ then } (A, \{\}) \text{ else } (\{\}, A))$
by (*simp add: btyping2-def btyping1-btyping2-aux*)

lemma *btyping2-aux-subset:*
 $\Vdash b (\subseteq A, X) = \text{Some } A' \implies A' = \{s. s \in A \wedge \text{bval } b \ s\}$
by (*induction b arbitrary: A', auto split: if-split-asm option.split-asm*)

lemma *btyping2-aux-diff:*
 $\Vdash b (\subseteq A, X) = \text{Some } B; \Vdash b (\subseteq A', X') = \text{Some } B'; A' \subseteq A; B' \subseteq B \implies$
 $A' - B' \subseteq A - B$
by (*blast dest: btyping2-aux-subset*)

lemma *btyping2-aux-mono:*
 $\Vdash b (\subseteq A, X) = \text{Some } B; A' \subseteq A; X \subseteq X' \implies$
 $\exists B'. \Vdash b (\subseteq A', X') = \text{Some } B' \wedge B' \subseteq B$
by (*induction b arbitrary: B, auto dest: btyping2-aux-diff split: if-split-asm option.split-asm*)

lemma *btyping2-mono:*
 $\Vdash b (\subseteq A, X) = (B_1, B_2); \Vdash b (\subseteq A', X') = (B_1', B_2'); A' \subseteq A; X \subseteq X' \implies$
 $B_1' \subseteq B_1 \wedge B_2' \subseteq B_2$
by (*simp add: btyping2-def split: option.split-asm, frule-tac [3-4] btyping2-aux-mono, auto dest: btyping2-aux-subset*)

lemma *btyping2-un-eq:*
 $\Vdash b (\subseteq A, X) = (B_1, B_2) \implies B_1 \cup B_2 = A$
by (*auto simp: btyping2-def dest: btyping2-aux-subset split: option.split-asm*)

lemma *btyping2-aux-eq:*
 $\Vdash b (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X) \implies \text{bval } b \ s = \text{bval } b \ t$
proof (*induction b arbitrary: A'*)
fix $A' \ v$
show
 $\Vdash Bc \ v (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X) \implies$
 $\text{bval } (Bc \ v) \ s = \text{bval } (Bc \ v) \ t$

```

    by simp
next
fix A' b
show
[[ $\wedge A'. \models b (\subseteq A, X) = \text{Some } A' \implies s = t (\subseteq \text{state} \cap X) \implies$ 
   $\text{bval } b \ s = \text{bval } b \ t;$ 
 $\models \text{Not } b (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X)$ ]]  $\implies$ 
   $\text{bval } (\text{Not } b) \ s = \text{bval } (\text{Not } b) \ t$ 
by (simp split: option.split-asm)
next
fix A' b1 b2
show
[[ $\wedge A'. \models b_1 (\subseteq A, X) = \text{Some } A' \implies s = t (\subseteq \text{state} \cap X) \implies$ 
   $\text{bval } b_1 \ s = \text{bval } b_1 \ t;$ 
 $\wedge A'. \models b_2 (\subseteq A, X) = \text{Some } A' \implies s = t (\subseteq \text{state} \cap X) \implies$ 
   $\text{bval } b_2 \ s = \text{bval } b_2 \ t;$ 
 $\models \text{And } b_1 \ b_2 (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X)$ ]]  $\implies$ 
   $\text{bval } (\text{And } b_1 \ b_2) \ s = \text{bval } (\text{And } b_1 \ b_2) \ t$ 
by (simp split: option.split-asm)
next
fix A' a1 a2
show
[[ $\models \text{Less } a_1 \ a_2 (\subseteq A, X) = \text{Some } A'; s = t (\subseteq \text{state} \cap X)$ ]]  $\implies$ 
   $\text{bval } (\text{Less } a_1 \ a_2) \ s = \text{bval } (\text{Less } a_1 \ a_2) \ t$ 
by (subgoal-tac aval a1 s = aval a1 t,
  subgoal-tac aval a2 s = aval a2 t,
  auto intro!: avars-aval split: if-split-asm)
qed

```

lemma *ctyping1-mono-fst*:
 $\vdash c (\subseteq A, X) = (B, Y); \vdash c (\subseteq A', X') = (B', Y'); A' \subseteq A \implies$
 $B' \subseteq B$
 by (fastforce simp: ctyping1-def)

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 (rule conjI, rule ctyping1-mono-fst [OF A B C])
 {
 fix x
 assume $x \notin \text{Univ}?? A' \{x. \forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in \vdash c\}.$
 if $f x = []$ then $x \in X'$ else $\text{snd } (\text{last } (f x)) \neq \text{None}$
 moreover from this have $A' \neq \{\}$
 by (simp split: if-split-asm)

ultimately have $\neg (\forall f.$
 $(\exists ys. f = (\lambda x. [y \leftarrow ys. fst\ y = x]) \wedge ys \in \vdash c) \longrightarrow$
 $(if\ f\ x = []\ then\ x \in X' \ else\ snd\ (last\ (f\ x)) \neq None))$
 $(is\ \neg\ ?P\ X')$
by (*auto split: if-split-asm*)
moreover assume $?P\ X$
hence $?P\ X'$
using D **by** *fastforce*
ultimately have *False*
by *contradiction*
 $\}$
with A **and** B **and** C **show** $Y \subseteq Y'$
by (*cases* $A = \{\}$), *auto simp: ctying1-def*)
qed

lemma *ctying2-mono-skip* [*elim!*]:
 $\llbracket (U, False) \models SKIP (\subseteq A, X) = Some\ (C, Z); A' \subseteq A; X \subseteq X' \rrbracket \implies$
 $\exists C' Z'. (U', False) \models SKIP (\subseteq A', X') = Some\ (C', Z') \wedge$
 $C' \subseteq C \wedge Z \subseteq Z'$
by (*clarsimp, subgoal-tac Univ?? C X = X, force+*)

lemma *ctying2-mono-assign* [*elim!*]:
 $\llbracket (U, False) \models x ::= a (\subseteq A, X) = Some\ (C, Z); A' \subseteq A; X \subseteq X';$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies$
 $\exists C' Z'. (U', False) \models x ::= a (\subseteq A', X') = Some\ (C', Z') \wedge$
 $C' \subseteq C \wedge Z \subseteq Z'$
by (*frule interf-set-mono [where W = {x}], auto split: if-split-asm*)

lemma *ctying2-mono-input* [*elim!*]:
 $\llbracket (U, False) \models IN\ x (\subseteq A, X) = Some\ (C, Z); A' \subseteq A; X \subseteq X';$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies$
 $\exists C' Z'. (U', False) \models IN\ x (\subseteq A', X') = Some\ (C', Z') \wedge$
 $C' \subseteq C \wedge Z \subseteq Z'$
by (*frule interf-set-mono [where W = {x}], auto split: if-split-asm*)

lemma *ctying2-mono-output* [*elim!*]:
 $\llbracket (U, False) \models OUT\ x (\subseteq A, X) = Some\ (C, Z); A' \subseteq A; X \subseteq X';$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies$
 $\exists C' Z'. (U', False) \models OUT\ x (\subseteq A', X') = Some\ (C', Z') \wedge$
 $C' \subseteq C \wedge Z \subseteq Z'$
by (*frule interf-set-mono [where W = {x}], auto split: if-split-asm*)

lemma *ctying2-mono-seq*:
assumes
 $A: \bigwedge A' B X' Y U'.$
 $(U, False) \models c_1 (\subseteq A, X) = Some\ (B, Y) \implies A' \subseteq A \implies X \subseteq X' \implies$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$
 $\exists B' Y'. (U', False) \models c_1 (\subseteq A', X') = Some\ (B', Y') \wedge$

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

proof –

obtain $B Y$ **where** $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \wedge$
 $(U, \text{False}) \models c_2 (\subseteq B, Y) = \text{Some } (C, Z)$
using C **by** (*auto split: option.split-asm*)
moreover from this obtain $B' Y'$ **where**
 $G: (U', \text{False}) \models c_1 (\subseteq A', X') = \text{Some } (B', Y') \wedge B' \subseteq B \wedge Y \subseteq Y'$
using A **and** D **and** E **and** F **by** *fastforce*
ultimately obtain $C' Z'$ **where**
 $(U', \text{False}) \models c_2 (\subseteq B', Y') = \text{Some } (C', Z') \wedge C' \subseteq C \wedge Z \subseteq Z'$
using B **and** F **by** *fastforce*
thus *?thesis*
using G **by** *simp*

qed

lemma *ctyping2-mono-or*:

assumes

A: $\bigwedge A' C_1 X' Y_1 U'$
 $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C_1, Y_1) \implies A' \subseteq A \implies X \subseteq X' \implies$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$
 $\exists C_1' Y_1'. (U', \text{False}) \models c_1 (\subseteq A', X') = \text{Some } (C_1', Y_1') \wedge$
 $C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1'$ **and**
B: $\bigwedge A' C_2 X' Y_2 U'$
 $(U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C_2, Y_2) \implies A' \subseteq A \implies X \subseteq X' \implies$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \implies$
 $\exists C_2' Y_2'. (U', \text{False}) \models c_2 (\subseteq A', X') = \text{Some } (C_2', Y_2') \wedge$
 $C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2'$ **and**
C: $(U, \text{False}) \models c_1$ **OR** $c_2 (\subseteq A, X) = \text{Some } (C, Y)$ **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 $\exists C' Y'. (U', \text{False}) \models c_1$ **OR** $c_2 (\subseteq A', X') = \text{Some } (C', Y') \wedge$
 $C' \subseteq C \wedge Y \subseteq Y'$

proof –

obtain $C_1 C_2 Y_1 Y_2$ **where**

$G: (C, Y) = (C_1 \cup C_2, Y_1 \cap Y_2) \wedge$
 $\text{Some } (C_1, Y_1) = (U, \text{False}) \models c_1 (\subseteq A, X) \wedge$

Some $(C_2, Y_2) = (U, \text{False}) \models c_2 (\subseteq A, X)$
using C by (*simp split: option.split-asm prod.split-asm*)
moreover have $H: \forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y$
using F by *simp*
ultimately have $\exists C_1' Y_1'$.
 $(U', \text{False}) \models c_1 (\subseteq A', X') = \text{Some} (C_1', Y_1') \wedge C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1'$
using A and D and E by *simp*
moreover have $\exists C_2' Y_2'$.
 $(U', \text{False}) \models c_2 (\subseteq A', X') = \text{Some} (C_2', Y_2') \wedge C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2'$
using B and D and E and G and H by *simp*
ultimately show *?thesis*
using G by *auto*
qed

lemma *ctyping2-mono-if*:

assumes

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

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

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

proof –

let $?W = \text{insert} (\text{Univ? } A X, \text{bvars } b) U$

let $?W' = \text{insert} (\text{Univ? } A' X', \text{bvars } b) U'$

obtain $B_1 B_2 C_1 C_2 Y_1 Y_2$ **where**

$G: (C, Y) = (C_1 \cup C_2, Y_1 \cap Y_2) \wedge (B_1, B_2) = \models b (\subseteq A, X) \wedge$

$\text{Some} (C_1, Y_1) = (?W, \text{False}) \models c_1 (\subseteq B_1, X) \wedge$

$\text{Some} (C_2, Y_2) = (?W, \text{False}) \models c_2 (\subseteq B_2, X)$

using C by (*simp split: option.split-asm prod.split-asm*)

moreover obtain $B_1' B_2'$ **where** $H: (B_1', B_2') = \models b (\subseteq A', X')$

by (*cases* $\models b (\subseteq A', X')$, *simp*)

ultimately have $I: B_1' \subseteq B_1 \wedge B_2' \subseteq B_2$

by (*metis btyping2-mono D E*)

moreover have $J: \forall (B', Y') \in ?W'. \exists (B, Y) \in ?W. B' \subseteq B \wedge Y' \subseteq Y$

using D and E and F by (*auto simp: univ-states-if-def*)

ultimately have $\exists C_1' Y_1'$.

$(?W', False) \models c_1 (\subseteq B_1', X') = Some (C_1', Y_1') \wedge C_1' \subseteq C_1 \wedge Y_1 \subseteq Y_1'$
using A and E and G by force
moreover have $\exists C_2' Y_2'$.
 $(?W', False) \models c_2 (\subseteq B_2', X') = Some (C_2', Y_2') \wedge C_2' \subseteq C_2 \wedge Y_2 \subseteq Y_2'$
using B and E and G and I and J by force
ultimately show *?thesis*
using G and H by (auto split: prod.split)
qed

lemma *ctyping2-mono-while*:

assumes

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

B: $\bigwedge B_1 B_2 C Y B_1' B_2' D_1' F Y' W U'. (B_1, B_2) = \models b (\subseteq A, X) \implies$
 $(C, Y) = \vdash c (\subseteq B_1, X) \implies (B_1', B_2') = \models b (\subseteq C, Y) \implies$
 $\forall (B, W) \in insert (Univ? A X \cup Univ? C Y, bvars b) U.$
B: $W \rightsquigarrow UNIV \implies$
 $(\{\}, False) \models c (\subseteq B_1', Y) = Some (F, W) \implies D_1' \subseteq B_1' \implies$
 $Y \subseteq Y' \implies \forall (B', Y') \in U'. \exists (B, Y) \in \{\}. B' \subseteq B \wedge Y' \subseteq Y \implies$
 $\exists F' W'. (U', False) \models c (\subseteq D_1', Y') = Some (F', W') \wedge$
 $F' \subseteq F \wedge W \subseteq W'$ **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 –

obtain $B_1 B_1' B_2 B_2' C E F V W Y$ where $G: (B_1, B_2) = \models b (\subseteq A, X) \wedge$
 $(C, Y) = \vdash c (\subseteq B_1, X) \wedge (B_1', B_2') = \models b (\subseteq C, Y) \wedge$
 $(\forall (B, W) \in insert (Univ? A X \cup Univ? C Y, bvars b) U. B: W \rightsquigarrow UNIV) \wedge$
 $Some (E, V) = (\{\}, False) \models c (\subseteq B_1, X) \wedge$
 $Some (F, W) = (\{\}, False) \models c (\subseteq B_1', Y) \wedge$
 $(B, Z) = (B_2 \cup B_2', Univ?? B_2 X \cap Y)$
using C by (force split: if-split-asm option.split-asm prod.split-asm)

moreover obtain $D_1 D_2$ where $H: \models b (\subseteq A', X') = (D_1, D_2)$

by (cases $\models b (\subseteq A', X')$, simp)

ultimately have $I: D_1 \subseteq B_1 \wedge D_2 \subseteq B_2$

by (smt (verit) btyping2-mono D E)

moreover obtain $C' Y'$ where $J: \vdash c (\subseteq D_1, X') = (C', Y')$

by (cases $\vdash c (\subseteq D_1, X')$, simp)

ultimately have $K: C' \subseteq C \wedge Y \subseteq Y'$

by (smt (verit) ctyping1-mono E G)

moreover obtain $D_1' D_2'$ **where** $L: \models b (\subseteq C', Y') = (D_1', D_2')$
 by (*cases* $\models b (\subseteq C', Y')$, *simp*)
ultimately have $M: D_1' \subseteq B_1' \wedge D_2' \subseteq B_2'$
 by (*smt* (*verit*) *btyping2-mono* G)
then obtain $F' W'$ **where**
 $(\{\}, False) \models c (\subseteq D_1', Y') = Some (F', W') \wedge F' \subseteq F \wedge W \subseteq W'$
 using B and F and G and K by *force*
moreover obtain $E' V'$ **where**
 $(\{\}, False) \models c (\subseteq D_1, X') = Some (E', V') \wedge E' \subseteq E \wedge V \subseteq V'$
 using A and E and F and G and I by *force*
moreover have $Univ? A' X' \subseteq Univ? A X$
 using D and E by (*auto simp: univ-states-if-def*)
moreover have $Univ? C' Y' \subseteq Univ? C Y$
 using K by (*auto simp: univ-states-if-def*)
ultimately have $(U', False) \models WHILE\ b\ DO\ c (\subseteq A', X') =$
 $Some (D_2 \cup D_2', Univ?? D_2 X' \cap Y')$
 using F and G and H and J and L by *force*
moreover have $D_2 \cup D_2' \subseteq B$
 using G and I and M by *auto*
moreover have $Z \subseteq Univ?? D_2 X' \cap Y'$
 using E and G and I and K by *auto*
ultimately show *?thesis*
 by *simp*
qed

lemma *ctyping2-mono*:

$\llbracket (U, False) \models c (\subseteq A, X) = Some (C, Z); A' \subseteq A; X \subseteq X';$
 $\forall (B', Y') \in U'. \exists (B, Y) \in U. B' \subseteq B \wedge Y' \subseteq Y \rrbracket \implies$
 $\exists C' Z'. (U', False) \models c (\subseteq A', X') = Some (C', Z') \wedge C' \subseteq C \wedge Z \subseteq Z'$
apply (*induction* $(U, False) c A X$ *arbitrary: A' C X' Z U U'*)
rule: ctyping2.induct
apply *fastforce*
apply *fastforce*
apply *fastforce*
apply *fastforce*
apply (*erule ctyping2-mono-seq, assumption+*)
apply (*erule ctyping2-mono-or, assumption+*)
apply (*erule ctyping2-mono-if, assumption+*)
apply (*erule ctyping2-mono-while, assumption+*)
done

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 –
 let $?F = \lambda x' w. \text{if } x = x'$

```

then (x, w) # [y←[]. fst y = x]
else [y←[]. fst y = x]
{
  fix s'
  assume s' ∈ C'
  moreover assume x ∈ state and C: avars a = {}
  ultimately obtain s where D: s ∈ A and E: s' = s(x := aval a s)
  using B by (auto split: if-split-asm)
  have ∃ s.
    (∃ t. s' = (λx'. if ?F x' (Some (aval a (λx. 0))) = []
      then s x'
      else case snd (last (?F x' (Some (aval a (λx. 0)))) of
        None ⇒ t x' | Some i ⇒ i) ∧
      s ∈ A
    apply (insert C E)
    apply (rule exI [of - s])
    apply (rule conjI [OF - D])
    apply (rule exI [of - λx. 0])
    by (fastforce intro: avars-aval)
  }
  moreover {
    fix s'
    assume s' ∈ C'
    moreover assume x ∈ state and avars a ≠ {}
    ultimately obtain s where C: s ∈ A and D: s' = s
    using B by (simp split: if-split-asm)
    have ∃ s.
      (∃ t. s' = (λx'. if ?F x' None = []
        then s x'
        else case snd (last (?F x' None)) of
          None ⇒ t x' | Some i ⇒ i) ∧
      s ∈ A
    apply (insert D)
    apply (rule exI [of - s])
    apply (rule conjI [OF - C])
    apply (rule exI [of - s])
    by auto
  }
  moreover {
    fix s'
    assume s' ∈ C' and x ∉ state
    hence s' ∈ A
    using B by (simp split: if-split-asm)
  }
  ultimately show ?thesis
  using A by (fastforce simp: ctyping1-def)
qed

```

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 –
let $?F = \lambda x'. \text{if } x = x'$
 $\text{then } (x, None) \# [y \leftarrow \square]. \text{fst } y = x'$
 $\text{else } [y \leftarrow \square]. \text{fst } y = x'$
{
fix s'
assume $s' \in C'$
moreover assume $x \in \text{state}$
ultimately obtain s **where** $C: s \in A$ **and** $D: s' = s$
using B **by** (*simp split: if-split-asm*)
have $\exists s.$
 $(\exists t. s' = (\lambda x'. \text{if } ?F\ x' = \square$
 $\text{then } s\ x'$
 $\text{else case } \text{snd } (\text{last } (?F\ x')) \text{ of}$
 $\text{None } \Rightarrow t\ x' \mid \text{Some } i \Rightarrow i)) \wedge$
 $s \in A$
apply (*insert D*)
apply (*rule exI [of - s]*)
apply (*rule conjI [OF - C]*)
apply (*rule exI [of - s]*)
by auto
}
moreover {
fix s'
assume $s' \in C'$ **and** $x \notin \text{state}$
hence $s' \in A$
using B **by** (*simp split: if-split-asm*)
}
ultimately show $?thesis$
using A **by** (*fastforce simp: ctyping1-def*)
qed

lemma *ctyping1-ctyping2-fst-output* [*elim!*]:
 $\llbracket \vdash OUT\ x (\subseteq A, X) = (C, Z);$
 $(U, False) \models OUT\ x (\subseteq A, X) = Some\ (C', Z') \rrbracket \Longrightarrow$
 $C' \subseteq C$
by (*simp add: ctyping1-def split: if-split-asm*)

lemma *ctyping1-ctyping2-fst-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**
 $C: \bigwedge B\ B'\ Y\ Y'. \vdash c_1 (\subseteq A, X) = (B, Y) \Longrightarrow$
 $(U, False) \models c_1 (\subseteq A, X) = Some\ (B', Y') \Longrightarrow B' \subseteq B$ **and**
 $D: \bigwedge p\ B'\ Y'\ D'\ C'\ W'\ Z'.$

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

proof –

obtain $B' Y'$ **where** $E: (U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (B', Y')$ **and**
 $(U, \text{False}) \models c_2 (\subseteq B', Y') = \text{Some } (C', Z')$

using B **by** (*auto split: option.split-asm*)

moreover obtain $D' W'$ **where** $F: \vdash c_2 (\subseteq B', Y') = (D', W')$

by (*cases* $\vdash c_2 (\subseteq B', Y')$, *simp*)

ultimately have $G: C' \subseteq D'$

using D **by** *simp*

obtain $B Y$ **where** $H: \vdash c_1 (\subseteq A, X) = (B, Y)$

by (*cases* $\vdash c_1 (\subseteq A, X)$, *simp*)

hence $B' \subseteq B$

using C **and** E **by** *simp*

moreover obtain $D W$ **where** $I: \vdash c_2 (\subseteq B, Y) = (D, W)$

by (*cases* $\vdash c_2 (\subseteq B, Y)$, *simp*)

ultimately have $D' \subseteq D$

using F **by** (*blast dest: ctyping1-mono-fst*)

moreover {

fix $ys\ ys'\ s\ t$ **and** $t' :: \text{state}$

assume $K: s \in A$

assume $ys \in \vdash c_1$ **and** $ys' \in \vdash c_2$

hence $L: ys \ @\ ys' \in \vdash c_1 \sqcup_{@} \vdash c_2$

by (*force simp: ctyping1-merge-append-def*
ctyping1-append-def ctyping1-merge-def)

let $?f = \lambda x. [y \leftarrow ys \ @\ ys'. \text{fst } y = x]$

let $?t = \lambda x. \text{if } [y \leftarrow ys'. \text{fst } y = x] = [] \text{ then } t\ x \text{ else } t'\ x$

have $\exists f\ s'$.

$(\exists t''.$

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

$\text{ then } s\ x$

$\text{ else case } \text{snd } (\text{last } [y \leftarrow ys. \text{fst } y = x]) \text{ of}$

$\text{ None } \Rightarrow t\ x \mid \text{ Some } i \Rightarrow i$

$\text{ else case } \text{snd } (\text{last } [y \leftarrow ys'. \text{fst } y = x]) \text{ of}$

$\text{ None } \Rightarrow t'\ x \mid \text{ Some } i \Rightarrow i) =$

$(\lambda x. \text{if } f\ x = []$

$\text{ then } s'\ x$

$\text{ else case } \text{snd } (\text{last } (f\ x)) \text{ of } \text{None } \Rightarrow t''\ x \mid \text{Some } i \Rightarrow i)) \wedge$

$(\exists ys''. f = (\lambda x. [y \leftarrow ys''. \text{fst } y = x]) \wedge ys'' \in \vdash c_1 \sqcup_{@} \vdash c_2) \wedge s' \in A$

apply (*insert K L*)

apply (*rule exI [of - ?f]*)

apply (*rule exI [of - s]*)

apply (*rule conjI*)

apply (*rule exI [of - ?t]*)

apply *fastforce*

apply (*rule conjI*)

```

    apply (rule exI [of - ys @ ys'])
    by simp-all
  }
  hence  $D \subseteq C$ 
    using  $A$  and  $H$  and  $I$  by (auto simp: ctyping1-def)
  ultimately show ?thesis
    using  $G$  by simp
qed

lemma ctyping1-ctyping2-fst-or:
  assumes
     $A: \vdash c_1 \text{ OR } c_2 (\subseteq A, X) = (C, Y)$  and
     $B: (U, \text{False}) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C', Y')$  and
     $C: \bigwedge C' Y' Y'. \vdash c_1 (\subseteq A, X) = (C, Y) \implies$ 
       $(U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C', Y') \implies C' \subseteq C$  and
     $D: \bigwedge C' Y' Y'. \vdash c_2 (\subseteq A, X) = (C, Y) \implies$ 
       $(U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C', Y') \implies C' \subseteq C$ 
  shows  $C' \subseteq C$ 
  proof -
    obtain  $C_1' C_2' Y_1' Y_2'$  where
       $E: (C', Y') = (C_1' \cup C_2', Y_1' \cap Y_2')$  and
       $F: (U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C_1', Y_1')$  and
       $G: (U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C_2', Y_2')$ 
      using  $B$  by (auto split: option.split-asm prod.split-asm)
    obtain  $C_1 Y_1$  where  $H: \vdash c_1 (\subseteq A, X) = (C_1, Y_1)$ 
      by (cases  $\vdash c_1 (\subseteq A, X)$ , simp)
    hence  $C_1' \subseteq C_1$ 
      using  $C$  and  $F$  by simp
    moreover obtain  $C_2 Y_2$  where  $I: \vdash c_2 (\subseteq A, X) = (C_2, Y_2)$ 
      by (cases  $\vdash c_2 (\subseteq A, X)$ , simp)
    hence  $C_2' \subseteq C_2$ 
      using  $D$  and  $G$  by simp
    ultimately have  $C' \subseteq C_1 \cup C_2$ 
      using  $E$  by blast
    moreover {
      fix  $ys\ s\ t$ 
      assume  $s \in A$ 
      moreover assume  $ys \in \vdash c_1$ 
      hence  $ys \in \vdash c_1 \sqcup \vdash c_2$ 
        by (force simp: ctyping1-merge-def)
      ultimately have  $\exists f\ s'$ .
        ( $\exists t'$ .
          ( $\lambda x.$  if  $[y \leftarrow ys. \text{fst } y = x] = []$ 
            then  $s\ x$ 
            else case  $\text{snd } (\text{last } [y \leftarrow ys. \text{fst } y = x])$  of
              None  $\Rightarrow t\ x \mid \text{Some } i \Rightarrow i) =$ 
          ( $\lambda x.$  if  $f\ x = []$ 
            then  $s'\ x$ 
            else case  $\text{snd } (\text{last } (f\ x))$  of None  $\Rightarrow t'\ x \mid \text{Some } i \Rightarrow i) \wedge$ 

```

```

    (∃ ys'. f = (λx. [y←ys'. fst y = x]) ∧ ys' ∈ ⊢ c₁ ⊔ ⊢ c₂) ∧ s' ∈ A
  by fastforce
}
hence C₁ ⊆ C
  using A and H by (auto simp: ctyping1-def)
moreover {
  fix ys s t
  assume s ∈ A
  moreover assume ys ∈ ⊢ c₂
  hence ys ∈ ⊢ c₁ ⊔ ⊢ c₂
    by (force simp: ctyping1-merge-def)
  ultimately have ∃ f s'.
    (∃ t'.
      (λx. 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) =
      (λx. if f x = []
        then s' x
        else case snd (last (f x)) of None ⇒ t' x | Some i ⇒ i)) ∧
      (∃ ys'. f = (λx. [y←ys'. fst y = x]) ∧ ys' ∈ ⊢ c₁ ⊔ ⊢ c₂) ∧ s' ∈ A
    by fastforce
  }
hence C₂ ⊆ C
  using A and I by (auto simp: ctyping1-def)
ultimately show ?thesis
  by blast
qed

```

lemma *ctyping1-ctyping2-fst-if*:

assumes

A: ⊢ IF b THEN c₁ ELSE c₂ (⊆ A, X) = (C, Y) **and**

B: (U, False) ⊨ IF b THEN c₁ ELSE c₂ (⊆ A, X) = Some (C', Y') **and**

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

(U', p) = (insert (Univ? A X, bvars b) U, ⊨ b (⊆ A, X)) ⇒

(B₁, B₂) = p ⇒ ⊢ c₁ (⊆ B₁, X) = (C, Y) ⇒

(U', False) ⊨ c₁ (⊆ B₁, X) = Some (C', Y') ⇒ C' ⊆ C **and**

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

(U', p) = (insert (Univ? A X, bvars b) U, ⊨ b (⊆ A, X)) ⇒

(B₁, B₂) = p ⇒ ⊢ c₂ (⊆ B₂, X) = (C, Y) ⇒

(U', False) ⊨ c₂ (⊆ B₂, X) = Some (C', Y') ⇒ C' ⊆ C

shows C' ⊆ C

proof –

let ?U' = insert (Univ? A X, bvars b) U

obtain B₁ B₂ C₁' C₂' Y₁' Y₂' **where**

E: (C', Y') = (C₁' ∪ C₂', Y₁' ∩ Y₂') **and**

F: ⊨ b (⊆ A, X) = (B₁, B₂) **and**

G: (?U', False) ⊨ c₁ (⊆ B₁, X) = Some (C₁', Y₁') **and**

H: (?U', False) ⊨ c₂ (⊆ B₂, X) = Some (C₂', Y₂')

```

    using B by (auto split: option.split-asm prod.split-asm)
  obtain C1 Y1 where I: ⊢ c1 (⊆ B1, X) = (C1, Y1)
    by (cases ⊢ c1 (⊆ B1, X), simp)
  hence C'1 ⊆ C1
    using C and F and G by simp
  moreover obtain C2 Y2 where J: ⊢ c2 (⊆ B2, X) = (C2, Y2)
    by (cases ⊢ c2 (⊆ B2, X), simp)
  hence C'2 ⊆ C2
    using D and F and H by simp
  ultimately have K: C' ⊆ C1 ∪ C2
    using E by blast
{
  fix ys s t
  assume s ∈ B1
  hence s ∈ A
    using F by (blast dest: btyping2-un-eq)
  moreover assume ys ∈ ⊢ c1
  hence ys ∈ ⊢ c1 ∪ ⊢ c2
    by (force simp: ctyping1-merge-def)
  ultimately have ∃ f s'.
    (∃ t'.
      (λx. 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) =
      (λx. if f x = []
        then s' x
        else case snd (last (f x)) of None ⇒ t' x | Some i ⇒ i)) ∧
      (∃ ys'. f = (λx. [y←ys'. fst y = x]) ∧ ys' ∈ ⊢ c1 ∪ ⊢ c2) ∧ s' ∈ A
    ) by fastforce
}
}
moreover {
  fix ys s t
  assume s ∈ B1
  moreover assume ys ∈ ⊢ c1
  hence ys ∈ ⊢ c1 ∪ {}
    by (force simp: ctyping1-merge-def)
  ultimately have ∃ f s'.
    (∃ t'.
      (λx. 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) =
      (λx. if f x = []
        then s' x
        else case snd (last (f x)) of None ⇒ t' x | Some i ⇒ i)) ∧
      (∃ ys'. f = (λx. [y←ys'. fst y = x]) ∧ ys' ∈ ⊢ c1 ∪ {}) ∧ s' ∈ B1
    ) by fastforce
}
}

```

```

ultimately have L:  $C_1 \subseteq C$ 
  using A and F and I by (cases  $\vdash b$ , auto
    dest!: btyping1-btyping2 [of - - A X] simp: ctyping1-def)
{
  fix ys s t
  assume  $s \in B_2$ 
  hence  $s \in A$ 
    using F by (blast dest: btyping2-un-eq)
  moreover assume  $ys \in \vdash c_2$ 
  hence  $ys \in \vdash c_1 \sqcup \vdash c_2$ 
    by (force simp: ctyping1-merge-def)
  ultimately have  $\exists f s'$ .
    ( $\exists t'$ .
      ( $\lambda x$ . if  $[y \leftarrow ys]. \text{fst } y = x$ ] = []
        then  $s \ x$ 
        else case snd (last  $[y \leftarrow ys]. \text{fst } y = x$ ) of
          None  $\Rightarrow t \ x \mid$  Some  $i \Rightarrow i$ ) =
      ( $\lambda x$ . if  $f \ x = []$ 
        then  $s' \ x$ 
        else case snd (last (f x)) of None  $\Rightarrow t' \ x \mid$  Some  $i \Rightarrow i$ ))  $\wedge$ 
      ( $\exists ys'. f = (\lambda x. [y \leftarrow ys']. \text{fst } y = x)$ )  $\wedge$   $ys' \in \vdash c_1 \sqcup \vdash c_2$ )  $\wedge$   $s' \in A$ 
    by fastforce
}
}
moreover {
  fix ys s t
  assume  $s \in B_2$ 
  moreover assume  $ys \in \vdash c_2$ 
  hence  $ys \in \{\}$   $\sqcup \vdash c_2$ 
    by (force simp: ctyping1-merge-def)
  ultimately have  $\exists f s'$ .
    ( $\exists t'$ .
      ( $\lambda x$ . if  $[y \leftarrow ys]. \text{fst } y = x$ ] = []
        then  $s \ x$ 
        else case snd (last  $[y \leftarrow ys]. \text{fst } y = x$ ) of
          None  $\Rightarrow t \ x \mid$  Some  $i \Rightarrow i$ ) =
      ( $\lambda x$ . if  $f \ x = []$ 
        then  $s' \ x$ 
        else case snd (last (f x)) of None  $\Rightarrow t' \ x \mid$  Some  $i \Rightarrow i$ ))  $\wedge$ 
      ( $\exists ys'. f = (\lambda x. [y \leftarrow ys']. \text{fst } y = x)$ )  $\wedge$   $ys' \in \{\} \sqcup \vdash c_2$ )  $\wedge$   $s' \in B_2$ 
    by fastforce
}
}
ultimately have  $C_2 \subseteq C$ 
  using A and F and J by (cases  $\vdash b$ , auto
    dest!: btyping1-btyping2 [of - - A X] simp: ctyping1-def)
  with K and L show ?thesis
  by blast
qed

```

lemma ctyping1-ctyping2-fst-while:

assumes

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

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

shows $B' \subseteq B$

proof –

obtain $B_1 B_1' B_2 B_2' C Y$ **where**

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

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

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

$F: (B', Z') = (B_2 \cup B_2', \text{Univ}?? B_2 X \cap Y)$

using B **by** (*force split: if-split-asm option.split-asm prod.split-asm*)

{

fix s

assume $s \in B_2$

hence $s \in A$

using C **by** (*blast dest: btyping2-un-eq*)

hence $\exists f s'$.

$(\exists t. s = (\lambda x. \text{if } f x = []$

then $s' x$

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

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

by *force*

}

with A **and** C **have** $G: B_2 \subseteq B$

by (*cases* $\vdash b$, *auto dest!: btyping1-btyping2 [of - - A X]*

simp: ctyping1-def)

{

fix s

assume $s \in B_2'$

hence $s \in C$

using E **by** (*blast dest: btyping2-un-eq*)

then obtain $f s'$ **where** H :

$(\exists t. s = (\lambda x. \text{if } f x = []$

then $s' x$

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

$(\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \wedge s' \in B_1$

using D **by** (*fastforce simp: ctyping1-def*)

hence $I: s' \in A$

using C **by** (*blast dest: btyping2-un-eq*)

have $\exists f s'$.

$(\exists t. s = (\lambda x. \text{if } f x = []$

then $s' x$

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

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

by (*rule exI [of - f], insert H I, auto*)

}

moreover {

fix s

assume $s \in B_2'$

```

moreover assume  $\vdash b = \text{Some True}$ 
ultimately have  $\exists f s'$ .
  ( $\exists t. s = (\lambda x. \text{if } f x = []$ 
     $\text{ then } s' x$ 
     $\text{ else case snd (last (f x)) of None } \Rightarrow t x \mid \text{Some } i \Rightarrow i)$ )  $\wedge$ 
  ( $\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c$ )  $\wedge s' \in A$ 
  using  $E$  by (auto dest: btyping1-btyping2 [of - - C Y])
}
moreover {
  fix  $s$ 
  assume  $s \in B_2'$ 
  hence  $C \neq \{\}$ 
  using  $E$  by (blast dest: btyping2-un-eq)
  hence  $B_1 \neq \{\}$ 
  using  $D$  by (auto simp: ctyping1-def)
  moreover assume  $\vdash b = \text{Some False}$ 
  ultimately have  $s \in A$ 
  using  $C$  by (auto dest: btyping1-btyping2 [of - - A X])
}
ultimately have  $B_2' \subseteq B$ 
using  $A$  by (cases  $\vdash b$ , auto simp: ctyping1-def)
with  $F$  and  $G$  show ?thesis
by simp
qed

```

```

lemma ctyping1-ctyping2-fst:
 $\llbracket \vdash c (\subseteq A, X) = (C, Z); (U, \text{False}) \models c (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \Longrightarrow$ 
 $C' \subseteq C$ 
apply (induction (U, False) c A X arbitrary: C C' Z Z' U)
rule: ctyping2.induct)
  apply (fastforce simp: ctyping1-def)
  apply fastforce
  apply fastforce
  apply fastforce
  apply (erule ctyping1-ctyping2-fst-seq, assumption+)
  apply (erule ctyping1-ctyping2-fst-or, assumption+)
  apply (erule ctyping1-ctyping2-fst-if, assumption+)
  apply (erule ctyping1-ctyping2-fst-while, assumption+)
done

```

```

lemma ctyping1-ctyping2-snd-skip [elim!]:
 $\llbracket \vdash \text{SKIP } (\subseteq A, X) = (C, Z);$ 
 $(U, \text{False}) \models \text{SKIP } (\subseteq A, X) = \text{Some } (C', Z') \rrbracket \Longrightarrow$ 
 $Z \subseteq Z'$ 
by (simp add: ctyping1-def split: if-split-asm)

```

```

lemma ctyping1-ctyping2-snd-assign [elim!]:
 $\llbracket \vdash x ::= a (\subseteq A, X) = (C, Z);$ 

```

$(U, \text{False}) \models x ::= a (\subseteq A, X) = \text{Some } (C', Z') \implies Z \subseteq Z'$

by (*auto simp: ctyping1-def split: if-split-asm*)

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

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

by (*auto simp: ctyping1-def split: if-split-asm*)

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

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

by (*simp add: ctyping1-def split: if-split-asm*)

lemma *ctyping1-ctyping2-snd-seq*:

assumes

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

shows $Z \subseteq Z'$

proof –

obtain $B' Y'$ **where** $E: (U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (B', Y')$ **and**
 $(U, \text{False}) \models c_2 (\subseteq B', Y') = \text{Some } (C', Z')$

using B **by** (*auto split: option.split-asm*)

moreover obtain $D' W'$ **where** $F: \vdash c_2 (\subseteq B', Y') = (D', W')$

by (*cases* $\vdash c_2 (\subseteq B', Y')$, *simp*)

ultimately have $G: W' \subseteq Z'$

using D **by** *simp*

obtain $B Y$ **where** $H: \vdash c_1 (\subseteq A, X) = (B, Y)$

by (*cases* $\vdash c_1 (\subseteq A, X)$, *simp*)

hence $Y \subseteq Y'$

using C **and** E **by** *simp*

moreover have $B' \subseteq B$

using H **and** E **by** (*rule ctyping1-ctyping2-fst*)

moreover obtain $D W$ **where** $I: \vdash c_2 (\subseteq B, Y) = (D, W)$

by (*cases* $\vdash c_2 (\subseteq B, Y)$, *simp*)

ultimately have $W \subseteq W'$

using F **by** (*blast dest: ctyping1-mono*)

moreover {

fix x

assume $J: \forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c_1 \sqcup_{\text{@}} \vdash c_2) \longrightarrow (if\ f\ x = []\ then\ x \in X\ else\ \text{snd } (\text{last } (f\ x)) \neq \text{None})$

```

{
  fix ys' ys
  assume ys ∈ ⊢ c1 and ys' ∈ ⊢ c2
  hence ys @ ys' ∈ ⊢ c1 ⊔@ ⊢ c2
  by (force simp: ctyping1-merge-append-def
      ctyping1-append-def ctyping1-merge-def)
  moreover assume [y←ys. fst y = x] = [] and [y←ys'. fst y = x] = []
  ultimately have x ∈ X
  using J by auto
}
moreover {
  fix ys ys'
  assume ys ∈ ⊢ c1 and ys' ∈ ⊢ c2
  hence ys @ ys' ∈ ⊢ c1 ⊔@ ⊢ c2
  by (force simp: ctyping1-merge-append-def
      ctyping1-append-def ctyping1-merge-def)
  moreover assume [y←ys. fst y = x] ≠ [] and [y←ys'. fst y = x] = []
  ultimately have ∃ i. snd (last [y←ys. fst y = x]) = Some i
  using J by auto
}
moreover {
  fix ys'
  assume ys' ∈ ⊢ c2
  moreover obtain ys where ys ∈ ⊢ c1
  by (insert ctyping1-aux-nonempty, blast)
  ultimately have ys @ ys' ∈ ⊢ c1 ⊔@ ⊢ c2
  by (force simp: ctyping1-merge-append-def
      ctyping1-append-def ctyping1-merge-def)
  moreover assume [y←ys'. fst y = x] ≠ []
  ultimately have ∃ i. snd (last [y←ys'. fst y = x]) = Some i
  using J by auto
}
ultimately have x ∈ {x. ∀ f ∈ {λx. [y←ys. fst y = x] | ys. ys ∈ ⊢ c2}.
  if f x = []
  then x ∈ {x. ∀ f. (∃ ys. f = (λx. [y←ys. fst y = x]) ∧ ys ∈ ⊢ c1) →
    (if f x = [] then x ∈ X else snd (last (f x)) ≠ None)}
  else snd (last (f x)) ≠ None}
  (is - ∈ ?X)
  by auto
moreover assume x ∉ (if ∀ x f s.
  (∀ t. x ≠ (λx. if f x = [] then s x else case snd (last (f x)) of
    None ⇒ t x | Some i ⇒ i)) ∨
  (∀ ys. f = (λx. [y←ys. fst y = x]) → ys ∉ ⊢ c1) ∨ s ∉ A
  then UNIV else ?X)
hence x ∉ ?X
  by (auto split: if-split-asm)
ultimately have False
  by contradiction
}

```

hence $Z \subseteq W$
 using A and H and I by (*cases* $A = \{\}$, *auto simp: ctyping1-def*)
 ultimately show *?thesis*
 using G by *simp*
 qed

lemma *ctyping1-ctyping2-snd-or*:

assumes

$A: \vdash c_1 \text{ OR } c_2 (\subseteq A, X) = (C, Y)$ **and**

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

$C: \bigwedge C' C'' Y Y'. \vdash c_1 (\subseteq A, X) = (C, Y) \implies$

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

$D: \bigwedge C' C'' Y Y'. \vdash c_2 (\subseteq A, X) = (C, Y) \implies$

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

shows $Y \subseteq Y'$

proof –

obtain $C_1' C_2' Y_1' Y_2'$ **where**

$E: (C', Y') = (C_1' \cup C_2', Y_1' \cap Y_2')$ **and**

$F: (U, \text{False}) \models c_1 (\subseteq A, X) = \text{Some } (C_1', Y_1')$ **and**

$G: (U, \text{False}) \models c_2 (\subseteq A, X) = \text{Some } (C_2', Y_2')$

using B by (*auto split: option.split-asm prod.split-asm*)

obtain $C_1 Y_1$ **where** $H: \vdash c_1 (\subseteq A, X) = (C_1, Y_1)$

by (*cases* $\vdash c_1 (\subseteq A, X)$, *simp*)

hence $Y_1 \subseteq Y_1'$

using C and F by *simp*

moreover obtain $C_2 Y_2$ **where** $I: \vdash c_2 (\subseteq A, X) = (C_2, Y_2)$

by (*cases* $\vdash c_2 (\subseteq A, X)$, *simp*)

hence $Y_2 \subseteq Y_2'$

using D and G by *simp*

ultimately have $Y_1 \cap Y_2 \subseteq Y'$

using E by *blast*

moreover {

fix $x \ y_s$

assume $\forall f. (\exists y_s. f = (\lambda x. [y \leftarrow y_s. \text{fst } y = x]) \wedge y_s \in \vdash c_1 \sqcup \vdash c_2) \longrightarrow$

(*if* $f \ x = []$ *then* $x \in X$ *else* $\text{snd } (\text{last } (f \ x)) \neq \text{None}$)

moreover assume $y_s \in \vdash c_1$

hence $y_s \in \vdash c_1 \sqcup \vdash c_2$

by (*force simp: ctyping1-merge-def*)

ultimately have *if* $[y \leftarrow y_s. \text{fst } y = x] = []$

then $x \in X$ *else* $\text{snd } (\text{last } [y \leftarrow y_s. \text{fst } y = x]) \neq \text{None}$

(*is ?P*)

by *blast*

moreover assume $\neg ?P$

ultimately have *False*

by *contradiction*

}

hence $Y \subseteq Y_1$

using A and H by (*cases* $A = \{\}$, *auto simp: ctyping1-def*)

moreover {

fix $x\ ys$
assume $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. fst\ y = x]) \wedge ys \in \vdash\ c_1 \sqcup \vdash\ c_2) \longrightarrow$
(if $f\ x = []$ then $x \in X$ else $snd\ (last\ (f\ x)) \neq None$)
moreover assume $ys \in \vdash\ c_2$
hence $ys \in \vdash\ c_1 \sqcup \vdash\ c_2$
by *(force simp: ctyping1-merge-def)*
ultimately have *if $[y \leftarrow ys. fst\ y = x] = []$*
then $x \in X$ else $snd\ (last\ [y \leftarrow ys. fst\ y = x]) \neq None$
(is ?P)
by blast
moreover assume $\neg\ ?P$
ultimately have *False*
by contradiction
}
hence $Y \subseteq Y_2$
using A **and** I **by** *(cases $A = \{\}$, auto simp: ctyping1-def)*
ultimately show *?thesis*
by blast
qed

lemma *ctyping1-ctyping2-snd-if:*

assumes

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

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

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

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

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

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

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

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

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

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

shows $Y \subseteq Y'$

proof –

let $?U' = insert\ (Univ?\ A\ X,\ bvars\ b)\ U$

obtain $B_1\ B_2\ C_1'\ C_2'\ Y_1'\ Y_2'$ **where**

$E: (C',\ Y') = (C_1' \cup C_2',\ Y_1' \cap Y_2')$ **and**

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

$G: (?U', False) \models c_1\ (\subseteq\ B_1,\ X) = Some\ (C_1',\ Y_1')$ **and**

$H: (?U', False) \models c_2\ (\subseteq\ B_2,\ X) = Some\ (C_2',\ Y_2')$

using B **by** *(auto split: option.split-asm prod.split-asm)*

obtain $C_1\ Y_1$ **where** $I: \vdash\ c_1\ (\subseteq\ B_1,\ X) = (C_1,\ Y_1)$

by *(cases $\vdash\ c_1\ (\subseteq\ B_1,\ X)$, simp)*

hence $Y_1 \subseteq Y_1'$

using C **and** F **and** G **by** *simp*

moreover obtain $C_2\ Y_2$ **where** $J: \vdash\ c_2\ (\subseteq\ B_2,\ X) = (C_2,\ Y_2)$

by *(cases $\vdash\ c_2\ (\subseteq\ B_2,\ X)$, simp)*

hence $Y_2 \subseteq Y_2'$

using D **and** F **and** H **by** *simp*

ultimately have $Y_1 \cap Y_2 \subseteq Y'$
using E *by blast*
moreover have $K: B_1 \cup B_2 = A$
using F *by (rule btyping2-un-eq)*
{
 fix $x x' ys$
 assume $x \in (if\ B_1 = \{\} \wedge B_2 = \{\} \text{ then } UNIV \text{ else } \{x. \forall f \in \{\lambda x. [y \leftarrow ys. fst\ y = x] \mid ys. ys \in \vdash\ c_1 \sqcup \vdash\ c_2\}. \text{ if } f\ x = [] \text{ then } x \in X \text{ else } snd\ (last\ (f\ x)) \neq None\})$ **and**
 $x' \in B_1$
 hence $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. fst\ y = x]) \wedge ys \in \vdash\ c_1 \sqcup \vdash\ c_2) \longrightarrow$
 $(if\ f\ x = [] \text{ then } x \in X \text{ else } snd\ (last\ (f\ x)) \neq None)$
 by *(auto split: if-split-asm)*
 moreover assume $ys \in \vdash\ c_1$
 hence $ys \in \vdash\ c_1 \sqcup \vdash\ c_2$
 by *(force simp: ctyping1-merge-def)*
 ultimately have *if* $[y \leftarrow ys. fst\ y = x] = []$
 $\text{ then } x \in X \text{ else } snd\ (last\ [y \leftarrow ys. fst\ y = x]) \neq None$
 (is ?P)
 by blast
 moreover assume $\neg\ ?P$
 ultimately have *False*
 by contradiction
}
note $L = this$
{
 fix $x x' ys v$
 assume $x \in (if\ B_1 = \{\} \wedge B_2 = \{\} \text{ then } UNIV \text{ else } \{x. \forall f \in \{\lambda x. [y \leftarrow ys. fst\ y = x] \mid ys. ys \in (if\ v \text{ then } \vdash\ c_1 \text{ else } \{\}) \sqcup (if\ \neg\ v \text{ then } \vdash\ c_2 \text{ else } \{\})\}. \text{ if } f\ x = [] \text{ then } x \in X \text{ else } snd\ (last\ (f\ x)) \neq None\})$
 moreover assume $M: x' \in B_1$ **and**
 $(if\ v \text{ then } (B_1 \cup B_2, \{\}) \text{ else } (\{\}, B_1 \cup B_2)) = (B_1, B_2)$
 hence v
 by *(simp split: if-split-asm)*
 ultimately have
 $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. fst\ y = x]) \wedge ys \in \vdash\ c_1 \sqcup \{\}) \longrightarrow$
 $(if\ f\ x = [] \text{ then } x \in X \text{ else } snd\ (last\ (f\ x)) \neq None)$
 using M **by** *(auto split: if-split-asm)*
 moreover assume $ys \in \vdash\ c_1$
 hence $ys \in \vdash\ c_1 \sqcup \{\}$
 by *(force simp: ctyping1-merge-def)*
 ultimately have *if* $[y \leftarrow ys. fst\ y = x] = []$
 $\text{ then } x \in X \text{ else } snd\ (last\ [y \leftarrow ys. fst\ y = x]) \neq None$
 (is ?P)
 by blast
 moreover assume $\neg\ ?P$
 ultimately have *False*
 by contradiction
}

```

}
note  $M = this$ 
from  $A$  and  $F$  and  $I$  and  $K$  have  $Y \subseteq Y_1$ 
  apply (cases  $B_1 = \{\}$ )
  apply (fastforce simp: ctyping1-def)
  apply (cases  $\vdash b$ )
  by (auto dest!: btyping1-btyping2 [of - - A X] L M simp: ctyping1-def)
moreover {
  fix  $x x' ys$ 
  assume  $x \in (if B_1 = \{\} \wedge B_2 = \{\} then UNIV else$ 
     $\{x. \forall f \in \{\lambda x. [y \leftarrow ys. fst y = x] \mid ys. ys \in \vdash c_1 \sqcup \vdash c_2\}.$ 
     $if f x = [] then x \in X else snd (last (f x)) \neq None\}$  and
     $x' \in B_2$ 
  hence  $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. fst y = x]) \wedge ys \in \vdash c_1 \sqcup \vdash c_2) \longrightarrow$ 
    (if  $f x = []$  then  $x \in X$  else  $snd (last (f x)) \neq None$ )
    by (auto split: if-split-asm)
  moreover assume  $ys \in \vdash c_2$ 
  hence  $ys \in \vdash c_1 \sqcup \vdash c_2$ 
    by (force simp: ctyping1-merge-def)
  ultimately have if  $[y \leftarrow ys. fst y = x] = []$ 
    then  $x \in X$  else  $snd (last [y \leftarrow ys. fst y = x]) \neq None$ 
    (is  $?P$ )
    by blast
  moreover assume  $\neg ?P$ 
  ultimately have False
    by contradiction
}
}
note  $N = this$ 
{
  fix  $x x' ys v$ 
  assume  $x \in (if B_1 = \{\} \wedge B_2 = \{\} then UNIV else$ 
     $\{x. \forall f \in \{\lambda x. [y \leftarrow ys. fst y = x] \mid ys.$ 
     $ys \in (if v then \vdash c_1 else \{\}) \sqcup (if \neg v then \vdash c_2 else \{\})\}.$ 
     $if f x = [] then x \in X else snd (last (f x)) \neq None\}$ 
  moreover assume  $O: x' \in B_2$  and
    (if  $v$  then  $(B_1 \cup B_2, \{\})$  else  $(\{\}, B_1 \cup B_2)$ ) =  $(B_1, B_2)$ 
  hence  $\neg v$ 
    by (simp split: if-split-asm)
  ultimately have
     $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. fst y = x]) \wedge ys \in \{\} \sqcup \vdash c_2) \longrightarrow$ 
    (if  $f x = []$  then  $x \in X$  else  $snd (last (f x)) \neq None$ )
    using  $O$  by (auto split: if-split-asm)
  moreover assume  $ys \in \vdash c_2$ 
  hence  $ys \in \{\} \sqcup \vdash c_2$ 
    by (force simp: ctyping1-merge-def)
  ultimately have if  $[y \leftarrow ys. fst y = x] = []$ 
    then  $x \in X$  else  $snd (last [y \leftarrow ys. fst y = x]) \neq None$ 
    (is  $?P$ )
    by blast
}

```

```

    moreover assume  $\neg ?P$ 
    ultimately have False
      by contradiction
  }
  note  $O = \text{this}$ 
  from  $A$  and  $F$  and  $J$  and  $K$  have  $Y \subseteq Y_2$ 
    apply (cases  $B_2 = \{\}$ )
    apply (fastforce simp: ctyping1-def)
    apply (cases  $\vdash b$ )
    by (auto dest!: btyping1-btyping2 [of - -  $A X$ ] NO simp: ctyping1-def)
  ultimately show ?thesis
    by blast
qed

```

lemma *ctyping1-ctyping2-snd-while*:

```

assumes
  A:  $\vdash \text{WHILE } b \text{ DO } c (\subseteq A, X) = (B, Z)$  and
  B:  $(U, \text{False}) \models \text{WHILE } b \text{ DO } c (\subseteq A, X) = \text{Some } (B', Z')$ 
shows  $Z \subseteq Z'$ 
proof -
  obtain  $B_1 B_1' B_2 B_2' C Y$  where
    C:  $\models b (\subseteq A, X) = (B_1, B_2)$  and
    D:  $\vdash c (\subseteq B_1, X) = (C, Y)$  and
    E:  $\models b (\subseteq C, Y) = (B_1', B_2')$  and
    F:  $(B', Z') = (B_2 \cup B_2', \text{Univ}?? B_2 X \cap Y)$ 
  using B by (force split: if-split-asm option.split-asm prod.split-asm)
  have  $G: B_1 \cup B_2 = A$ 
  using C by (rule btyping2-un-eq)
  {
    fix  $x x'$ 
    assume  $x \in (\text{if } B_1 = \{\} \wedge B_2 = \{\} \text{ then UNIV else } \{x. \forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys = [] \vee ys \in \vdash c\}. \text{if } f x = [] \text{ then } x \in X \text{ else } \text{snd } (\text{last } (f x)) \neq \text{None}\})$  and
       $x' \in B_2$ 
    hence  $\forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys = [] \vee ys \in \vdash c\}. \text{if } f x = [] \text{ then } x \in X \text{ else } \text{snd } (\text{last } (f x)) \neq \text{None}$ 
      by (auto split: if-split-asm)
    hence  $x \in X$ 
      by fastforce
    moreover assume  $x \notin X$ 
    ultimately have False
      by contradiction
  }
  note  $H = \text{this}$ 
  {
    fix  $x x' v$ 
    assume  $x \in (\text{if } B_1 = \{\} \wedge B_2 = \{\} \text{ then UNIV else } \{x. \forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in (\text{if } \neg v \text{ then } \{\} \text{ else } \{\}) \vee ys \in (\text{if } v \text{ then } \vdash c \text{ else } \{\})\})$ .
  }

```

if $f x = []$ then $x \in X$ else $\text{snd } (\text{last } (f x)) \neq \text{None}$

moreover assume $H: x' \in B_2$ **and**
 (if v then $(B_1 \cup B_2, \{\})$ else $(\{\}, B_1 \cup B_2)$) = (B_1, B_2)

hence $\neg v$
 by (simp split: if-split-asm)

ultimately have $x \in X$
 using H by (auto split: if-split-asm)

moreover assume $x \notin X$
ultimately have *False*
 by contradiction

}

note $I = \text{this}$

from A **and** C **and** G **have** $Z \subseteq \text{Univ}?? B_2 X$

apply (cases $B_2 = \{\}$)
apply fastforce
apply (cases $\vdash b$)
 by (auto dest!: btyping1-btyping2 [of - - $A X$] $H I$ simp: ctyping1-def)

moreover {

fix x
assume $x \notin \text{Univ}?? B_1 \{x. \forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in \vdash c\}$.
 if $f x = []$ then $x \in X$ else $\text{snd } (\text{last } (f x)) \neq \text{None}$

moreover from this have $B_1 \neq \{\}$
 by (simp split: if-split-asm)

ultimately have $\neg (\forall f.$
 $(\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge (ys = [] \vee ys \in \vdash c)) \longrightarrow$
 $(\text{if } f x = [] \text{ then } x \in X \text{ else } \text{snd } (\text{last } (f x)) \neq \text{None}))$
 (is $\neg ?P$)
 by (auto split: if-split-asm)

moreover assume $?P$
ultimately have *False*
 by contradiction

}

note $J = \text{this}$

{

fix $x v$
assume $x \notin \text{Univ}?? B_1 \{x. \forall f \in \{\lambda x. [y \leftarrow ys. \text{fst } y = x] \mid ys. ys \in \vdash c\}$.
 if $f x = []$ then $x \in X$ else $\text{snd } (\text{last } (f x)) \neq \text{None}$

moreover from this have $K: B_1 \neq \{\}$
 by (simp split: if-split-asm)

ultimately have $L: \neg (\forall f.$
 $(\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge ys \in \vdash c) \longrightarrow$
 $(\text{if } f x = [] \text{ then } x \in X \text{ else } \text{snd } (\text{last } (f x)) \neq \text{None}))$
 (is $\neg ?P$)
 by (auto split: if-split-asm)

assume $\vdash b = \text{Some } v$

with C **and** K **have** v
 by (auto dest: btyping1-btyping2 [of - - $A X$])

moreover assume $\forall f. (\exists ys. f = (\lambda x. [y \leftarrow ys. \text{fst } y = x]) \wedge$
 $(ys \in (\text{if } \neg v \text{ then } \{\}\} \text{ else } \{\}) \vee ys \in (\text{if } v \text{ then } \vdash c \text{ else } \{\})) \longrightarrow$

```

      (if f x = [] then x ∈ X else snd (last (f x)) ≠ None)
    ultimately have ?P
      by simp
    with L have False
      by contradiction
  }
  note K = this
  from A and D and G have Z ⊆ Y
    apply (cases A = {})
    apply (fastforce simp: ctyping1-def)
    apply (cases ⊢ b)
    by (auto dest: J K simp: ctyping1-def)
  ultimately show ?thesis
    using F by simp
qed

```

lemma *ctyping1-ctyping2-snd*:

```

[[⊢ c (⊆ A, X) = (C, Z); (U, False) ⊨ c (⊆ A, X) = Some (C', Z')]] ⇒
  Z ⊆ Z'
apply (induction (U, False) c A X arbitrary: C C' Z Z' U
  rule: ctyping2.induct)
  apply fastforce
  apply fastforce
  apply fastforce
  apply fastforce
  apply (erule ctyping1-ctyping2-snd-seq, assumption+)
  apply (erule ctyping1-ctyping2-snd-or, assumption+)
  apply (erule ctyping1-ctyping2-snd-if, assumption+)
  apply (erule ctyping1-ctyping2-snd-while, assumption+)
done

```

lemma *ctyping1-ctyping2*:

```

[[⊢ c (⊆ A, X) = (C, Z); (U, False) ⊨ c (⊆ A, X) = Some (C', Z')]] ⇒
  C' ⊆ C ∧ Z ⊆ Z'
by (blast dest: ctyping1-ctyping2-fst ctyping1-ctyping2-snd)

```

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

```

assumes
  A: ⊨ b1 (⊆ A, X) = Some B1 and
  B: ⊨ b2 (⊆ A, X) = Some B2 and
  C: bval b1 s and
  D: bval b2 s and
  E: r ∈ A and
  F: s = r (⊆ state ∩ X)
shows ∃ r' ∈ B1 ∩ B2. r = r' (⊆ state ∩ X)
proof –
  from A and C and E and F have r ∈ B1

```

by (*frule-tac btyping2-aux-subset*, *drule-tac btyping2-aux-eq*, *auto*)
 moreover from B and D and E and F have $r \in B_2$
 by (*frule-tac btyping2-aux-subset*, *drule-tac btyping2-aux-eq*, *auto*)
 ultimately show *?thesis*
 by *blast*
 qed

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

assumes
 A : *avars* $a_1 \subseteq \text{state}$ and
 B : *avars* $a_2 \subseteq \text{state}$ and
 C : *avars* $a_1 \subseteq X$ and
 D : *avars* $a_2 \subseteq X$ and
 E : *aval* $a_1 s < \text{aval } a_2 s$ and
 F : $r \in A$ and
 G : $s = r (\subseteq \text{state} \cap X)$
 shows $\exists r'. r' \in A \wedge \text{aval } a_1 r' < \text{aval } a_2 r' \wedge r = r' (\subseteq \text{state} \cap X)$
 proof –
 have $\text{aval } a_1 s = \text{aval } a_1 r \wedge \text{aval } a_2 s = \text{aval } a_2 r$
 using A and B and C and D and G by (*blast intro: avars-aval*)
 thus *?thesis*
 using E and F by *auto*
 qed

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

assumes
 A : *avars* $a_1 \subseteq \text{state}$ and
 B : *avars* $a_2 \subseteq \text{state}$ and
 C : *avars* $a_1 \subseteq X$ and
 D : *avars* $a_2 \subseteq X$ and
 E : $\neg \text{aval } a_1 s < \text{aval } a_2 s$ and
 F : $r \in A$ and
 G : $s = r (\subseteq \text{state} \cap X)$
 shows $\exists r' \in A - \{s \in A. \text{aval } a_1 s < \text{aval } a_2 s\}. r = r' (\subseteq \text{state} \cap X)$
 proof –
 have $\text{aval } a_1 s = \text{aval } a_1 r \wedge \text{aval } a_2 s = \text{aval } a_2 r$
 using A and B and C and D and G by (*blast intro: avars-aval*)
 thus *?thesis*
 using E and F by *auto*
 qed

lemma *btyping2-aux-approx*:

$\llbracket \models b (\subseteq A, X) = \text{Some } A'; s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $s \in \text{Univ (if } \text{bval } b s \text{ then } A' \text{ else } A - A') (\subseteq \text{state} \cap X)$
 by (*induction b arbitrary: A'*, *auto dest: btyping2-aux-subset*
split: if-split-asm option.split-asm)

lemma *btyping2-approx*:

$\llbracket \models b (\subseteq A, X) = (B_1, B_2); s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$

$s \in \text{Univ}$ (if $\text{bval } b \ s$ then B_1 else B_2) ($\subseteq \text{state} \cap X$)
by (drule *sym*, *simp add: btyping2-def split: option.split-asm*,
drule *btyping2-aux-approx, auto*)

lemma *btyping2-approx-assign* [elim]:
 $\llbracket \forall t'. \text{aval } a \ s = t' \ x \longrightarrow (\forall s. t' = s(x := \text{aval } a \ s) \longrightarrow s \notin A) \vee$
 $(\exists y \in \text{state} \cap X. y \neq x \wedge t \neq t' \ y);$
 $v \models a \ (\subseteq X); t \in A; s = t \ (\subseteq \text{state} \cap X) \rrbracket \Longrightarrow \text{False}$
by (drule *spec [of - t(x := aval a t)]*, *cases a*,
(fastforce simp del: aval.simps(3) intro: avars-aval)+)

lemma *btyping2-approx-if-1*:
 $\llbracket \text{bval } b \ s; \models b \ (\subseteq A, X) = (B_1, B_2); r \in A; s = r \ (\subseteq \text{state} \cap X);$
 $(\text{insert } (\text{Univ? } A \ X, \text{bvars } b) \ U, v) \models c_1 \ (\subseteq B_1, X) = \text{Some } (C_1, Y_1);$
 $\bigwedge A \ B \ X \ Y \ U \ v. (U, v) \models c_1 \ (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists r \in A. s = r \ (\subseteq \text{state} \cap X) \Longrightarrow \exists r' \in B. t = r' \ (\subseteq \text{state} \cap Y) \rrbracket \Longrightarrow$
 $\exists r' \in C_1 \cup C_2. t = r' \ (\subseteq \text{state} \cap (Y_1 \cap Y_2))$
by (drule *btyping2-approx, blast, fastforce*)

lemma *btyping2-approx-if-2*:
 $\llbracket \neg \text{bval } b \ s; \models b \ (\subseteq A, X) = (B_1, B_2); r \in A; s = r \ (\subseteq \text{state} \cap X);$
 $(\text{insert } (\text{Univ? } A \ X, \text{bvars } b) \ U, v) \models c_2 \ (\subseteq B_2, X) = \text{Some } (C_2, Y_2);$
 $\bigwedge A \ B \ X \ Y \ U \ v. (U, v) \models c_2 \ (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists r \in A. s = r \ (\subseteq \text{state} \cap X) \Longrightarrow \exists r' \in B. t = r' \ (\subseteq \text{state} \cap Y) \rrbracket \Longrightarrow$
 $\exists r' \in C_1 \cup C_2. t = r' \ (\subseteq \text{state} \cap (Y_1 \cap Y_2))$
by (drule *btyping2-approx, blast, fastforce*)

lemma *btyping2-approx-while-1* [elim]:
 $\llbracket \neg \text{bval } b \ s; r \in A; s = r \ (\subseteq \text{state} \cap X); \models b \ (\subseteq A, X) = (B, \{\}) \rrbracket \Longrightarrow$
 $\exists t \in C. s = t \ (\subseteq \text{state} \cap Y)$
by (drule *btyping2-approx, blast, simp*)

lemma *btyping2-approx-while-2* [elim]:
 $\llbracket \forall t \in B_2 \cup B_2'. \exists x \in \text{state} \cap (X \cap Y). r \ x \neq t \ x; \neg \text{bval } b \ s;$
 $r \in A; s = r \ (\subseteq \text{state} \cap X); \models b \ (\subseteq A, X) = (B_1, B_2) \rrbracket \Longrightarrow \text{False}$
by (drule *btyping2-approx, blast, auto*)

lemma *btyping2-approx-while-aux*:
assumes
 $A: \models b \ (\subseteq A, X) = (B_1, B_2)$ **and**
 $B: \vdash c \ (\subseteq B_1, X) = (C, Y)$ **and**
 $C: \models b \ (\subseteq C, Y) = (B_1', B_2')$ **and**
 $D: (\{\}, \text{False}) \models c \ (\subseteq B_1, X) = \text{Some } (D, Z)$ **and**
 $E: (\{\}, \text{False}) \models c \ (\subseteq B_1', Y) = \text{Some } (D', Z')$ **and**
 $F: r_1 \in A$ **and**
 $G: s_1 = r_1 \ (\subseteq \text{state} \cap X)$ **and**
 $H: \text{bval } b \ s_1$ **and**
 $I: \bigwedge C \ B \ Y \ W \ U. (\text{case } \models b \ (\subseteq C, Y) \text{ of } (B_1', B_2') \Rightarrow$

$\text{case } \vdash c (\subseteq B_1', Y) \text{ of } (C', Y') \Rightarrow$
 $\text{case } \models b (\subseteq C', Y') \text{ of } (B_1'', B_2'') \Rightarrow \text{if}$
 $(\forall s \in \text{Univ? } C Y \cup \text{Univ? } C' Y'. \forall x \in \text{bvars } b. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y) \wedge$
 $(\forall p \in U. \text{case } p \text{ of } (B, W) \Rightarrow \forall s \in B. \forall x \in W. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y)$
 $\text{then case } (\{\}, \text{False}) \models c (\subseteq B_1', Y) \text{ of}$
 $\text{None} \Rightarrow \text{None} \mid \text{Some } - \Rightarrow \text{case } (\{\}, \text{False}) \models c (\subseteq B_1'', Y') \text{ of}$
 $\text{None} \Rightarrow \text{None} \mid \text{Some } - \Rightarrow \text{Some } (B_2' \cup B_2'', \text{Univ?? } B_2' Y \cap Y')$
 $\text{else None} = \text{Some } (B, W) \Longrightarrow$
 $\exists r \in C. s_2 = r (\subseteq \text{state} \cap Y) \Longrightarrow \exists r \in B. s_3 = r (\subseteq \text{state} \cap W)$
 $(\text{is } \bigwedge C B Y W U. ?P C B Y W U \Longrightarrow - \Longrightarrow -) \text{ and}$
 $J: \bigwedge A B X Y U v. (U, v) \models c (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists r \in A. s_1 = r (\subseteq \text{state} \cap X) \Longrightarrow \exists r \in B. s_2 = r (\subseteq \text{state} \cap Y) \text{ and}$
 $K: \forall s \in \text{Univ? } A X \cup \text{Univ? } C Y. \forall x \in \text{bvars } b. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y \text{ and}$
 $L: \forall p \in U. \forall B W. p = (B, W) \longrightarrow$
 $(\forall s \in B. \forall x \in W. \forall y. s: \text{dom } x \rightsquigarrow \text{dom } y)$
 $\text{shows } \exists r \in B_2 \cup B_2'. s_3 = r (\subseteq \text{state} \cap \text{Univ?? } B_2 X \cap Y)$
proof –
obtain $C' Y'$ **where** $M: \vdash c (\subseteq B_1', Y) = (C', Y')$
by ($\text{cases } \vdash c (\subseteq B_1', Y)$, simp)
obtain $B_1'' B_2''$ **where** $N: (B_1'', B_2'') = \models b (\subseteq C', Y')$
by ($\text{cases } \models b (\subseteq C', Y')$, simp)
let $?B = B_2' \cup B_2''$
let $?W = \text{Univ?? } B_2' Y \cap Y'$
have $\vdash c (\subseteq C, Y) = (C, Y)$
using ctyping1-idem **and** B **by** auto
moreover have $B_1' \subseteq C$
using C **by** ($\text{blast dest: btyping2-un-eq}$)
ultimately have $O: C' \subseteq C \wedge Y \subseteq Y'$
by ($\text{rule ctyping1-mono [OF - M]}$, simp)
hence $\text{Univ? } C' Y' \subseteq \text{Univ? } C Y$
by ($\text{auto simp: univ-states-if-def}$)
moreover from I **have** $?P C ?B Y ?W U \Longrightarrow$
 $\exists r \in C. s_2 = r (\subseteq \text{state} \cap Y) \Longrightarrow \exists r \in ?B. s_3 = r (\subseteq \text{state} \cap ?W) .$
ultimately have ($\text{case } (\{\}, \text{False}) \models c (\subseteq B_1'', Y')$ **of**
 $\text{None} \Rightarrow \text{None} \mid \text{Some } - \Rightarrow \text{Some } (?B, ?W) = \text{Some } (?B, ?W) \Longrightarrow$
 $\exists r \in C. s_2 = r (\subseteq \text{state} \cap Y) \Longrightarrow \exists r \in ?B. s_3 = r (\subseteq \text{state} \cap ?W)$
using C **and** E **and** K **and** L **and** M **and** N
by ($\text{fastforce split: if-split-asm prod.split-asm}$)
moreover have $P: B_1'' \subseteq B_1' \wedge B_2'' \subseteq B_2'$
by ($\text{metis btyping2-mono C N O}$)
hence $\exists D'' Z''. (\{\}, \text{False}) \models c (\subseteq B_1'', Y') =$
 $\text{Some } (D'', Z'') \wedge D'' \subseteq D' \wedge Z' \subseteq Z''$
using E **and** O **by** ($\text{auto intro: ctyping2-mono}$)
ultimately have
 $\exists r \in C. s_2 = r (\subseteq \text{state} \cap Y) \Longrightarrow \exists r \in ?B. s_3 = r (\subseteq \text{state} \cap ?W)$
by fastforce
moreover from A **and** D **and** F **and** G **and** H **and** J **obtain** r_2 **where**
 $r_2 \in D$ **and** $s_2 = r_2 (\subseteq \text{state} \cap Z)$
by ($\text{drule-tac btyping2-approx, blast, force}$)

moreover have $D \subseteq C \wedge Y \subseteq Z$
using B and D **by** (rule *ctyping1-ctyping2*)
ultimately obtain r_3 **where** $Q: r_3 \in ?B$ **and** $R: s_3 = r_3 (\subseteq \text{state} \cap ?W)$
by *blast*
show *?thesis*
proof (rule *bezI* [*of - r3*])
show $s_3 = r_3 (\subseteq \text{state} \cap \text{Univ}?? B_2 X \cap Y)$
using O and R **by** *auto*
next
show $r_3 \in B_2 \cup B_2'$
using P and Q **by** *blast*
qed
qed

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

lemma *ctyping2-approx-while-4*:

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

by (drule *ctyping2-approx-while-aux*, *assumption+*, *auto*)

lemma *ctyping2-approx*:

$\llbracket (c, s, p) \Rightarrow (t, q); (U, v) \models c (\subseteq A, X) = \text{Some } (B, Y);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \Rightarrow t \in \text{Univ } B (\subseteq \text{state} \cap Y)$
proof (*induction* (c, s, p) (t, q) *arbitrary*: $A B X Y U v c s p t q$
rule: *big-step.induct*)
fix $A C X Z U v c_1 c_2 s p t q$ **and** $p' :: \text{stage}$
show

$\llbracket \bigwedge r q A B X Y U v. p' = (r, q) \implies$
 $(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies r \in \text{Univ } B (\subseteq \text{state} \cap Y);$
 $\bigwedge r q B C Y Z U v. p' = (r, q) \implies$
 $(U, v) \models c_2 (\subseteq B, Y) = \text{Some } (C, Z) \implies$
 $r \in \text{Univ } B (\subseteq \text{state} \cap Y) \implies t \in \text{Univ } C (\subseteq \text{state} \cap Z);$
 $(U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (C, Z);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $t \in \text{Univ } C (\subseteq \text{state} \cap Z)$
by (*cases p', auto split: option.split-asm prod.split-asm*)

next
fix $A C X Y U v c_1 c_2 s p t q$
show
 $\llbracket \bigwedge A C X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (C, Y) \implies$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies t \in \text{Univ } C (\subseteq \text{state} \cap Y);$
 $(U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C, Y);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $t \in \text{Univ } C (\subseteq \text{state} \cap Y)$
by (*fastforce split: option.split-asm*)

next
fix $A C X Y U v c_1 c_2 s p t q$
show
 $\llbracket \bigwedge A C X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (C, Y) \implies$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies t \in \text{Univ } C (\subseteq \text{state} \cap Y);$
 $(U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (C, Y);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $t \in \text{Univ } C (\subseteq \text{state} \cap Y)$
by (*fastforce split: option.split-asm*)

next
fix $A B X Y U v b c_1 c_2 s p t q$
show
 $\llbracket \text{bval } b s; (c_1, s, p) \Rightarrow (t, q);$
 $\bigwedge A C X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (C, Y) \implies$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies t \in \text{Univ } C (\subseteq \text{state} \cap Y);$
 $(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $t \in \text{Univ } B (\subseteq \text{state} \cap Y)$
by (*auto split: option.split-asm prod.split-asm,*
rule ctyping2-approx-if-1)

next
fix $A B X Y U v b c_1 c_2 s p t q$
show
 $\llbracket \neg \text{bval } b s; (c_2, s, p) \Rightarrow (t, q);$
 $\bigwedge A C X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (C, Y) \implies$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \implies t \in \text{Univ } C (\subseteq \text{state} \cap Y);$
 $(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $s \in \text{Univ } A (\subseteq \text{state} \cap X) \rrbracket \implies$
 $t \in \text{Univ } B (\subseteq \text{state} \cap Y)$
by (*auto split: option.split-asm prod.split-asm,*

```

    rule ctyping2-approx-if-2)
next
fix A B X Y U v b c s1 p1 s2 p2 s3 p3
show
  [[bval b s1; (c, s1, p1) ⇒ (s2, p2);
  ∧ A B X Y U v. (U, v) ⊨ c (⊆ A, X) = Some (B, Y) ⇒
  s1 ∈ Univ A (⊆ state ∩ X) ⇒ s2 ∈ Univ B (⊆ state ∩ Y);
  (WHILE b DO c, s2, p2) ⇒ (s3, p3);
  ∧ A B X Y U v. (U, v) ⊨ WHILE b DO c (⊆ A, X) = Some (B, Y) ⇒
  s2 ∈ Univ A (⊆ state ∩ X) ⇒ s3 ∈ Univ B (⊆ state ∩ Y);
  (U, v) ⊨ WHILE b DO c (⊆ A, X) = Some (B, Y);
  s1 ∈ Univ A (⊆ state ∩ X)] ⇒
  s3 ∈ Univ B (⊆ state ∩ Y)
by (auto split: if-split-asm option.split-asm prod.split-asm,
    erule-tac [2] ctyping2-approx-while-4,
    erule ctyping2-approx-while-3)
qed (auto split: if-split-asm option.split-asm prod.split-asm)

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 (⊆ bvars b) ⇒ bval b s = bval b t
by (induction b, simp-all, rule arg-cong2, auto intro: avars-aval)

```

```

lemma eq-streams-subset:
  [[f = f' (⊆ vs, vs', T); T' ⊆ T]] ⇒ f = f' (⊆ vs, vs', T')
by (auto simp: eq-streams-def)

```

```

lemma flow-append-1:
  assumes A: ∧ cfs' :: (com × stage) list.

```

$c \# \text{map fst } (cfs :: (\text{com} \times \text{stage}) \text{ list}) = \text{map fst } cfs' \implies$
 $\text{flow-aux } (\text{map fst } cfs' @ \text{map fst } cfs'') =$
 $\text{flow-aux } (\text{map fst } cfs') @ \text{flow-aux } (\text{map fst } cfs'')$
shows $\text{flow-aux } (c \# \text{map fst } cfs @ \text{map fst } cfs'') =$
 $\text{flow-aux } (c \# \text{map fst } cfs) @ \text{flow-aux } (\text{map fst } cfs'')$
using A [of $(c, \lambda x. 0, \lambda x n. 0, [], []) \# cfs$] **by** *simp*

lemma *flow-append*:

$\text{flow } (cfs @ cfs') = \text{flow } cfs @ \text{flow } cfs'$
by (*simp add: flow-def, induction map fst cfs arbitrary: cfs*
rule: flow-aux.induct, auto, rule flow-append-1)

lemma *flow-cons*:

$\text{flow } (cf \# cfs) = \text{flow-aux } (\text{fst } cf \# []) @ \text{flow } cfs$
by (*subgoal-tac cf \# cfs = [cf] @ cfs, simp only: flow-append,*
simp-all add: flow-def)

lemma *in-flow-length*:

$\text{length } [p \leftarrow \text{in-flow } cs \text{ vs } f. \text{fst } p = x] = \text{length } [c \leftarrow cs. c = \text{IN } x]$
by (*induction cs vs f rule: in-flow.induct, simp-all*)

lemma *in-flow-append*:

$\text{in-flow } (cs @ cs') \text{ vs } f =$
 $\text{in-flow } cs \text{ vs } f @ \text{in-flow } cs' (\text{vs } @ \text{in-flow } cs \text{ vs } f) f$
by (*induction cs' vs f rule: in-flow.induct,*
(simp only: append-assoc [symmetric] in-flow.simps,
simp add: in-flow-length ac-simps)+)

lemma *in-flow-one*:

$\text{in-flow } [c] \text{ vs } f = (\text{case } c \text{ of}$
 $\text{IN } x \Rightarrow [(x, f x (\text{length } [p \leftarrow \text{vs}. \text{fst } p = x]))] \mid - \Rightarrow [])$
by (*subst append-Nil [symmetric], cases c, simp-all only: in-flow.simps,*
simp-all)

lemma *run-flow-append*:

$\text{run-flow } (cs @ cs') \text{ vs } s f =$
 $\text{run-flow } cs' (\text{vs } @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f$
by (*induction cs' vs s f rule: run-flow.induct,*
(simp only: append-assoc [symmetric] run-flow.simps,
simp add: in-flow-length ac-simps)+)

lemma *run-flow-one*:

$\text{run-flow } [c] \text{ vs } s f = (\text{case } c \text{ of}$
 $x ::= a \Rightarrow s(x := \text{aval } a \ s) \mid$
 $\text{IN } x \Rightarrow s(x := f x (\text{length } [p \leftarrow \text{vs}. \text{fst } p = x])) \mid$
 $- \Rightarrow s)$
by (*subst append-Nil [symmetric], cases c, simp-all only: run-flow.simps,*

simp-all)

lemma *run-flow-observe*:

run-flow ($\langle X \rangle \# cs$) *vs s f* = *run-flow* *cs vs s f*

apply (*rule subst* [*of* ($\square @ \langle X \rangle$)] @ *cs* -
 $\lambda cs'. \text{run-flow } cs' \text{ vs } s \text{ f} = \text{run-flow } cs \text{ vs } s \text{ f}$])
apply *fastforce*

by (*subst run-flow-append, simp only: in-flow.simps run-flow.simps, simp*)

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

by (*induction cs' vs s f rule: out-flow.induct,*
(simp only: append-assoc [symmetric] out-flow.simps,
simp add: run-flow-append)+)

lemma *out-flow-one*:

out-flow [*c*] *vs s f* = (*case c of*
OUT x $\Rightarrow [(x, s \ x)] \mid - \Rightarrow []$)

by (*subst append-Nil [symmetric], cases c, simp-all only: out-flow.simps,*
simp-all)

lemma *no-upd-empty*:

no-upd *cs {}*

by (*induction cs {} :: vname set rule: no-upd.induct, simp-all*)

lemma *no-upd-append*:

no-upd (*cs @ cs'*) *X* = (*no-upd* *cs X* \wedge *no-upd* *cs' X*)

by (*induction cs X rule: no-upd.induct, simp-all*)

lemma *no-upd-in-flow*:

no-upd *cs X* \Longrightarrow [*p*←*in-flow* *cs vs f. fst p* \in *X*] = []

by (*induction cs vs f rule: in-flow.induct, simp-all add: no-upd-append*)

lemma *no-upd-run-flow*:

no-upd *cs X* \Longrightarrow *run-flow* *cs vs s f* = *s* (\subseteq *X*)

by (*induction cs vs s f rule: run-flow.induct, auto simp: Let-def no-upd-append*)

lemma *no-upd-out-flow*:

no-upd *cs X* \Longrightarrow [*p*←*out-flow* *cs vs s f. fst p* \in *X*] = []

by (*induction cs vs s f rule: out-flow.induct, simp-all add: no-upd-append*)

lemma *small-stepsl-append*:

$\llbracket cf \rightarrow^* \{cfs\} cf'; cf' \rightarrow^* \{cfs'\} cf'' \rrbracket \Longrightarrow cf \rightarrow^* \{cfs @ cfs'\} cf''$

by (*induction cf' cfs' cf'' rule: small-stepsl.induct, simp,*

simp only: append-assoc [symmetric] small-stepsl.simps)

lemma *small-step-stream:*

$(c, s, f, vs, ws) \rightarrow (c', p) \implies \exists s' vs' ws'. p = (s', f, vs', ws')$
by (*induction (c, s, f, vs, ws) (c', p) arbitrary: c s f vs ws c' p*
rule: small-step.induct, simp-all)

lemma *small-stepsl-stream:*

$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', p) \implies \exists s' vs' ws'. p = (s', f, vs', ws')$
by (*induction (c, s, f, vs, ws) cfs (c', p) arbitrary: c s f vs ws c' p*
rule: small-stepsl.induct, auto dest: small-step-stream)

lemma *small-steps-stepsl-1:*

$\exists cfs. cf \rightarrow^*\{cfs\} cf$
by (*rule exI [of - []], simp)*

lemma *small-steps-stepsl-2:*

$\llbracket cf \rightarrow cf'; cf' \rightarrow^*\{cfs\} cf'' \rrbracket \implies \exists cfs'. cf \rightarrow^*\{cfs'\} cf''$
by (*rule exI [of - [cf] @ cfs], rule small-stepsl-append,*
subst append-Nil [symmetric], simp only: small-stepsl.simps)

lemma *small-steps-stepsl:*

$cf \rightarrow^* cf' \implies \exists cfs. cf \rightarrow^*\{cfs\} cf'$
by (*induction cf cf' rule: star.induct, rule small-steps-stepsl-1,*
blast intro: small-steps-stepsl-2)

lemma *small-stepsl-steps:*

$cf \rightarrow^*\{cfs\} cf' \implies cf \rightarrow^* cf'$
by (*induction cf cfs cf' rule: small-stepsl.induct, auto intro: star-trans)*

lemma *small-steps-stream:*

$(c, s, f, vs, ws) \rightarrow^* (c', p) \implies \exists s' vs' ws'. p = (s', f, vs', ws')$
by (*blast dest: small-steps-stepsl intro: small-stepsl-stream)*

lemma *small-stepsl-cons-1:*

$cf \rightarrow^*\{[cf']\} cf'' \implies cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{\} cf'')$
by (*subst (asm) append-Nil [symmetric], simp only: small-stepsl.simps,*
cases cf'', simp)

lemma *small-stepsl-cons-2:*

$\llbracket cf \rightarrow^*\{cf' \# cfs\} cf'' \rrbracket \implies$
 $cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs\} cf'');$
 $cf \rightarrow^*\{cf' \# cfs @ [cf']\} cf'' \implies$
 $cf' = cf \wedge (\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs @ [cf']\} cf'')$
by (*simp only: append-Cons [symmetric], simp only: small-stepsl.simps, simp)*

lemma *small-stepsl-cons:*

$cf \rightarrow^*\{cf' \# cfs\} cf'' \implies$
 $cf' = cf \wedge$
 $(\exists cf'. cf \rightarrow cf' \wedge cf' \rightarrow^*\{cfs\} cf'')$
by (induction $cf \ cfs \ cf''$ rule: *small-stepsl.induct*,
erule small-stepsl-cons-1, rule *small-stepsl-cons-2*)

lemma *small-stepsl-skip*:
 $(SKIP, p) \rightarrow^*\{cfs\} cf \implies cf = (SKIP, p) \wedge flow \ cfs = []$
by (induction $(SKIP, p) \ cfs \ cf$ rule: *small-stepsl.induct*,
auto simp: flow-def)

lemma *small-stepsl-assign*:
 $(x ::= a, s, p) \rightarrow^*\{cfs\} cf \implies$
 $cf = (x ::= a, s, p) \wedge$
 $flow \ cfs = [] \vee$
 $cf = (SKIP, s(x := aval \ a \ s), p) \wedge$
 $flow \ cfs = [x ::= a]$
by (induction $(x ::= a :: com, s, p) \ cfs \ cf$ rule: *small-stepsl.induct*,
force simp: flow-def, *auto simp: flow-append*, *simp-all add: flow-def*)

lemma *small-stepsl-input*:
 $(IN \ x, s, f, vs, ws) \rightarrow^*\{cfs\} cf \implies$
 $cf = (IN \ x, s, f, vs, ws) \wedge$
 $flow \ cfs = [] \vee$
 $(let \ n = length \ [p \leftarrow vs. \ fst \ p = x]$
 $in \ cf = (SKIP, s(x := f \ x \ n), f, vs @ [(x, f \ x \ n)], ws) \wedge$
 $flow \ cfs = [IN \ x])$
by (induction $(IN \ x :: com, s, f, vs, ws) \ cfs \ cf$ rule:
small-stepsl.induct, *force simp: flow-def*, *auto simp: Let-def flow-append*,
simp-all add: flow-def)

lemma *small-stepsl-output*:
 $(OUT \ x, s, f, vs, ws) \rightarrow^*\{cfs\} cf \implies$
 $cf = (OUT \ x, s, f, vs, ws) \wedge$
 $flow \ cfs = [] \vee$
 $cf = (SKIP, s, f, vs, ws @ [(x, s \ x)]) \wedge$
 $flow \ cfs = [OUT \ x]$
by (induction $(OUT \ x :: com, s, f, vs, ws) \ cfs \ cf$ rule:
small-stepsl.induct, *force simp: flow-def*, *auto simp: flow-append*,
simp-all add: flow-def)

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

$$(c_1, p) \rightarrow^* \{cfs'\} (SKIP, p') \wedge$$

$$(c_2, p') \rightarrow^* \{cfs''\} (c, q) \wedge$$

$$flow \square = flow cfs' @ flow cfs''$$

by force

lemma *small-stepsl-seq-2*:

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

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 (*cases cf*)

fix $c' q'$

assume $C: cf = (c', q')$

moreover {

assume $D: (c', q') \rightarrow (c, q)$

assume

$(\exists c'' cfs'. ?P c' q' c'' cfs') \vee$

$(\exists p' cfs' cfs''. ?Q c' q' p' cfs' cfs'')$

hence *?thesis*

proof

assume $\exists c'' cfs'. ?P c' q' c'' cfs'$

then obtain c'' **and** cfs' **where**

$E: c' = c'';; c_2$ **and**

$F: (c_1, p) \rightarrow^* \{cfs'\} (c'', q')$ **and**

$G: flow cfs = flow cfs'$

by blast

hence $(c'';; c_2, q') \rightarrow (c, q)$

using D **by simp**

moreover {

assume

$H: c'' = SKIP$ **and**

```

    I: (c, q) = (c2, q')
  have ?U
  proof (rule exI [of - q'], rule exI [of - cfs'],
        rule exI [of - []])
    from C and E and F and G and H and I show
      length [] < length (cfs @ [cf]) ∧
      (c1, p) →*{cfs'} (SKIP, q') ∧
      (c2, q') →*{[]} (c, q) ∧
      flow (cfs @ [cf]) = flow cfs' @ flow []
    by (simp add: flow-append, simp add: flow-def)
  qed
}
moreover {
  fix d q''
  assume
    H: (c'', q') → (d, q'') and
    I: (c, q) = (d;; c2, q'')
  have ?T
  proof (rule exI [of - d],
        rule exI [of - cfs' @ [(c'', q')]])
    from C and E and F and G and H and I show
      c = d;; c2 ∧
      (c1, p) →*{cfs' @ [(c'', q')]} (d, q) ∧
      flow (cfs @ [cf]) = flow (cfs' @ [(c'', q')])
    by (simp add: flow-append, simp add: flow-def)
  qed
}
ultimately show ?thesis
  by blast
next
assume ∃ p' cfs' cfs''. ?Q c' q' p' cfs' cfs''
then obtain p' and cfs' and cfs'' where
  E: length cfs'' < length cfs and
  F: (c1, p) →*{cfs'} (SKIP, p') and
  G: (c2, p') →*{cfs''} (c', q') and
  H: flow cfs = flow cfs' @ flow cfs''
  by blast
show ?thesis
proof (rule disjI2, rule exI [of - p'], rule exI [of - cfs'],
      rule exI [of - cfs'' @ [(c', q')]])
  from C and D and E and F and G and H show
    length (cfs'' @ [(c', q')]) < length (cfs @ [cf]) ∧
    (c1, p) →*{cfs'} (SKIP, p') ∧
    (c2, p') →*{cfs'' @ [(c', q')]} (c, q) ∧
    flow (cfs @ [cf]) = flow cfs' @ flow (cfs'' @ [(c', q')])
  by (simp add: flow-append)
qed
qed
}

```

ultimately show *?thesis*
 using *A* and *B* by *simp*
 qed

lemma *small-stepsl-seq*:

$$(c_1;; c_2, p) \rightarrow^*\{cfs\} (c, q) \implies$$

$$(\exists c' cfs'. c = c';; c_2 \wedge$$

$$(c_1, p) \rightarrow^*\{cfs'\} (c', q) \wedge$$

$$flow\ cfs = flow\ cfs') \vee$$

$$(\exists p' cfs' cfs''. length\ cfs'' < length\ cfs \wedge$$

$$(c_1, p) \rightarrow^*\{cfs'\} (SKIP, p') \wedge (c_2, p') \rightarrow^*\{cfs''\} (c, q) \wedge$$

$$flow\ cfs = flow\ cfs' @ flow\ cfs'')$$

by (*induction* $(c_1;; c_2, p)$ *cfs* (c, q) *arbitrary*: $c_1\ c_2\ p\ c\ q$
rule: *small-stepsl.induct*, *erule small-stepsl-seq-1*,
rule small-stepsl-seq-2)

lemma *small-stepsl-or-1*:

assumes *A*: $(c_1\ OR\ c_2, p) \rightarrow^*\{cfs\} cf \implies$
 $cf = (c_1\ OR\ c_2, p) \wedge$
 $flow\ cfs = [] \vee$
 $(c_1, p) \rightarrow^*\{tl\ cfs\} cf \wedge$
 $flow\ cfs = flow\ (tl\ cfs) \vee$
 $(c_2, p) \rightarrow^*\{tl\ cfs\} cf \wedge$
 $flow\ cfs = flow\ (tl\ cfs)$
(is - $\implies ?P \vee ?Q \vee ?R$)

assumes *B*: $(c_1\ OR\ c_2, p) \rightarrow^*\{cfs @ [cf]\} cf'$

shows

$$cf' = (c_1\ OR\ c_2, p) \wedge$$

$$flow\ (cfs @ [cf]) = [] \vee$$

$$(c_1, p) \rightarrow^*\{tl\ (cfs @ [cf])\} cf' \wedge$$

$$flow\ (cfs @ [cf]) = flow\ (tl\ (cfs @ [cf])) \vee$$

$$(c_2, p) \rightarrow^*\{tl\ (cfs @ [cf])\} cf' \wedge$$

$$flow\ (cfs @ [cf]) = flow\ (tl\ (cfs @ [cf]))$$

(is - $\vee ?T$)

proof -

{

assume

C: $(c_1\ OR\ c_2, p) \rightarrow^*\{cfs\} cf$ **and**

D: $cf \rightarrow cf'$

assume $?P \vee ?Q \vee ?R$

hence $?T$

proof (*rule disjE*, *erule-tac [2] disjE*)

assume $?P$

moreover from this have $(c_1\ OR\ c_2, p) \rightarrow cf'$

using *D* by *simp*

ultimately show *?thesis*

using *C* by (*auto dest: small-stepsl-cons*

simp: tl-append flow-cons split: list.split)

```

next
  assume ?Q
  with C and D show ?thesis
    by (auto simp: tl-append flow-cons split: list.split)
next
  assume ?R
  with C and D show ?thesis
    by (auto simp: tl-append flow-cons split: list.split)
qed
}
with A and B show ?thesis
  by simp
qed

```

lemma *small-steps-l-or*:

```

(c1 OR c2, p) →*{cfs} cf ⇒
  cf = (c1 OR c2, p) ∧
  flow cfs = [] ∨
  (c1, p) →*{tl cfs} cf ∧
  flow cfs = flow (tl cfs) ∨
  (c2, p) →*{tl cfs} cf ∧
  flow cfs = flow (tl cfs)

```

by (*induction* (c1 OR c2, p) cfs cf *rule: small-steps-l.induct*,
force simp: flow-def, rule small-steps-l-or-1)

lemma *small-steps-l-if-1*:

```

assumes A: (IF b THEN c1 ELSE c2, s, p) →*{cfs} cf ⇒
  cf = (IF b THEN c1 ELSE c2, s, p) ∧
  flow cfs = [] ∨
  bval b s ∧ (c1, s, p) →*{tl cfs} cf ∧
  flow cfs = ⟨bvars b⟩ # flow (tl cfs) ∨
  ¬ bval b s ∧ (c2, s, p) →*{tl cfs} cf ∧
  flow cfs = ⟨bvars b⟩ # flow (tl cfs)
(is - ⇒ ?P ∨ ?Q ∨ ?R)

```

assumes B: (IF b THEN c1 ELSE c2, s, p) →*{cfs @ [cf]} cf'

shows

```

cf' = (IF b THEN c1 ELSE c2, s, p) ∧
  flow (cfs @ [cf]) = [] ∨
  bval b s ∧ (c1, s, p) →*{tl (cfs @ [cf])} cf' ∧
  flow (cfs @ [cf]) = ⟨bvars b⟩ # flow (tl (cfs @ [cf])) ∨
  ¬ bval b s ∧ (c2, s, p) →*{tl (cfs @ [cf])} cf' ∧
  flow (cfs @ [cf]) = ⟨bvars b⟩ # flow (tl (cfs @ [cf]))
(is - ∨ ?T)

```

proof –

```

{
  assume
    C: (IF b THEN c1 ELSE c2, s, p) →*{cfs} cf and
    D: cf → cf'

```

assume $?P \vee ?Q \vee ?R$
hence $?T$
proof (rule *disjE*, erule-tac [2] *disjE*)
assume $?P$
moreover from this have $(IF\ b\ THEN\ c_1\ ELSE\ c_2, s, p) \rightarrow cf'$
using D by *simp*
ultimately show $?thesis$
using C by (auto dest: *small-stepsl-cons*
simp: tl-append flow-cons split: list.split)
next
assume $?Q$
with D **show** $?thesis$
by (auto *simp: tl-append flow-cons split: list.split*)
next
assume $?R$
with D **show** $?thesis$
by (auto *simp: tl-append flow-cons split: list.split*)
qed
}
with A and B **show** $?thesis$
by *simp*
qed

lemma *small-stepsl-if*:

$(IF\ b\ THEN\ c_1\ ELSE\ c_2, s, p) \rightarrow*\{cfs\}\ cf \implies$
 $cf = (IF\ b\ THEN\ c_1\ ELSE\ c_2, s, p) \wedge$
 $flow\ cfs = [] \vee$
 $bval\ b\ s \wedge (c_1, s, p) \rightarrow*\{tl\ cfs\}\ cf \wedge$
 $flow\ cfs = \langle bvars\ b \rangle \# flow\ (tl\ cfs) \vee$
 $\neg bval\ b\ s \wedge (c_2, s, p) \rightarrow*\{tl\ cfs\}\ cf \wedge$
 $flow\ cfs = \langle bvars\ b \rangle \# flow\ (tl\ cfs)$
by (induction (IF b THEN c_1 ELSE c_2, s, p) *cfs cf rule*:
small-stepsl.induct, force simp: flow-def, rule small-stepsl-if-1)

lemma *small-stepsl-while-1*:

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

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

proof –

```

{
  assume
    C: (WHILE b DO c, s, p) →*{cfs} cf and
    D: cf → cf'
  assume ?P ∨ ?Q ∨ ?R
  hence ?T
  proof (rule disjE, erule-tac [2] disjE)
    assume ?P
    moreover from this have (WHILE b DO c, s, p) → cf'
      using D by simp
    ultimately show ?thesis
      using C by (auto dest: small-steps-cons
        simp: tl-append flow-cons split: list.split)
  next
    assume ?Q
    with D show ?thesis
      by (auto simp: tl-append flow-cons split: list.split)
  next
    assume ?R
    with D show ?thesis
      by blast
  qed
}
with A and B show ?thesis
  by simp
qed

```

lemma *small-steps-while*:

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

by (*induction* (WHILE b DO c, s, p) cfs cf rule: *small-steps.induct*,
force simp: flow-def, rule small-steps-while-1)

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

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

arbitrary: c c' s s' f f' vs vs' ws ws' rule: small-step.induct,
auto elim: ssubst)

lemma *small-steps-in-flow:*

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

by (*induction* (c, s, f, vs, ws) (c', s', f', vs', ws'))
arbitrary: c c' s s' f f' vs vs' ws ws' rule: star.induct,
auto intro: small-steps-in-flow-1)

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

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

by (*induction* (c, s, f, vs, ws) (c', s', f', vs', ws'))
arbitrary: c c' s s' f f' vs vs' ws ws' rule: small-step.induct,
auto elim: ssubst)

lemma *small-steps-out-flow:*

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

by (*induction* (c, s, f, vs, ws) (c', s', f', vs', ws'))
arbitrary: c c' s s' f f' vs vs' ws ws' rule: star.induct,
auto intro: small-steps-out-flow-1)

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

assumes

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

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

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

using *small-stepsl-stream [OF A] and B*

by (*induction* $[c']$ *arbitrary: c' c'' rule: flow-aux.induct,*
auto simp: flow-def in-flow-one)

lemma *small-stepsl-in-flow:*

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

by (*induction* (c, s, f, vs, ws) $cfs (c', s', f', vs', ws')$)
arbitrary: c' s' f' vs' ws' rule: small-stepsl.induct, simp add: flow-def,
auto intro: small-stepsl-in-flow-1 simp: flow-append in-flow-append)

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

assumes

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

$B: (c', \text{run-flow } (\text{flow cfs}) \text{ vs } s f, f', vs', ws') \rightarrow$
 $(c'', s'', f'', vs'', ws'')$
shows $s'' = \text{run-flow } (\text{flow } [(c', \text{run-flow } (\text{flow cfs}) \text{ vs } s f, f', vs', ws')])$
 $(vs \text{ @ in-flow } (\text{flow cfs}) \text{ vs } f) (\text{run-flow } (\text{flow cfs}) \text{ vs } s f) f$
using *small-stepsl-stream* [OF A] **and** *small-stepsl-in-flow* [OF A] **and** B
by (*induction* [c'] *arbitrary*: $c' c''$ *rule*: *flow-aux.induct*,
auto simp: *flow-def run-flow-one*)

lemma *small-stepsl-run-flow*:

$(c, s, f, vs, ws) \rightarrow^*\{cfs\} (c', s', f', vs', ws') \implies$
 $s' = \text{run-flow } (\text{flow cfs}) \text{ vs } s f$
by (*induction* $(c, s, f, vs, ws) cfs (c', s', f', vs', ws')$
arbitrary: $c' s' f' vs' ws'$ *rule*: *small-stepsl.induct*, *simp add*: *flow-def*,
auto intro: *small-stepsl-run-flow-1 simp*: *flow-append run-flow-append*)

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

assumes

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

shows $ws'' = ws \text{ @ } ws' \text{ @}$

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

using *small-stepsl-run-flow* [OF A] **and** B

by (*induction* [c'] *arbitrary*: $c' c''$ *rule*: *flow-aux.induct*,
auto simp: *flow-def out-flow-one*)

lemma *small-stepsl-out-flow*:

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

by (*induction* $(c, s, f, vs, ws) cfs (c', s', f', vs', ws')$

arbitrary: $c' s' f' vs' ws'$ *rule*: *small-stepsl.induct*, *simp add*: *flow-def*,
auto intro: *small-stepsl-out-flow-1 simp*: *flow-append out-flow-append*)

lemma *small-steps-inputs*:

assumes

$A: (c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_0, s_1, f, vs_1, ws_1)$ **and**
 $B: (c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$ **and**
 $C: (c, s', f', vs_0', ws_0') \rightarrow^* (c_0', s_1', f', vs_1', ws_1')$ **and**
 $D: (c_1', s_1', f', vs_1', ws_1') \rightarrow^* (c_2', s_2', f', vs_2', ws_2')$ **and**

$E: \text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0) \text{ vs}_1. P p] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0') \text{ vs}_1'. P p]$ **and**

$F: \text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) \text{ vs}_2. P p] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') \text{ vs}_2'. P p]$

shows $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0) \text{ vs}_2. P p] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_0') \text{ vs}_2'. P p]$

proof –

have $G: vs_1 = vs_0 \text{ @ drop } (\text{length } vs_0) \text{ vs}_1$

using *small-stepsl-steps* [OF A] **by** (*rule small-steps-in-flow*)
have $vs_2 = vs_1 @ \text{drop} (\text{length } vs_1) vs_2$
using *small-stepsl-steps* [OF B] **by** (*rule small-steps-in-flow*)
hence $H: vs_2 = vs_0 @ \text{drop} (\text{length } vs_0) vs_1 @ \text{drop} (\text{length } vs_1) vs_2$
by (*subst (asm) G, simp*)
have $I: vs_1' = vs_0' @ \text{drop} (\text{length } vs_0') vs_1'$
using *C* **by** (*rule small-steps-in-flow*)
have $vs_2' = vs_1' @ \text{drop} (\text{length } vs_1') vs_2'$
using *D* **by** (*rule small-steps-in-flow*)
hence $J: vs_2' = vs_0' @ \text{drop} (\text{length } vs_0') vs_1' @ \text{drop} (\text{length } vs_1') vs_2'$
by (*subst (asm) I, simp*)
from *E* **and** *F* **show** *?thesis*
by (*subst H, subst J, simp*)
qed

lemma *small-steps-outputs*:

assumes

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

proof –

have $G: ws_1 = ws_0 @ \text{drop} (\text{length } ws_0) ws_1$
using *small-stepsl-steps* [OF A] **by** (*rule small-steps-out-flow*)
have $ws_2 = ws_1 @ \text{drop} (\text{length } ws_1) ws_2$
using *small-stepsl-steps* [OF B] **by** (*rule small-steps-out-flow*)
hence $H: ws_2 = ws_0 @ \text{drop} (\text{length } ws_0) ws_1 @ \text{drop} (\text{length } ws_1) ws_2$
by (*subst (asm) G, simp*)
have $I: ws_1' = ws_0' @ \text{drop} (\text{length } ws_0') ws_1'$
using *C* **by** (*rule small-steps-out-flow*)
have $ws_2' = ws_1' @ \text{drop} (\text{length } ws_1') ws_2'$
using *D* **by** (*rule small-steps-out-flow*)
hence $J: ws_2' = ws_0' @ \text{drop} (\text{length } ws_0') ws_1' @ \text{drop} (\text{length } ws_1') ws_2'$
by (*subst (asm) I, simp*)
from *E* **and** *F* **show** *?thesis*
by (*subst H, subst J, simp*)
qed

5.2 Local context proofs

context *noninterf*
begin

lemma *no-upd-sources*:

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

by (*induction cs rule: rev-induct, auto simp: no-upd-append split: com-flow.split*)

lemma *sources-aux-append*:

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

by (*induction cs' rule: rev-induct, simp, subst append-assoc [symmetric], auto simp del: append-assoc split: com-flow.split*)

lemma *sources-out-append*:

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

by (*induction cs' rule: rev-induct, simp, subst append-assoc [symmetric], auto simp del: append-assoc split: com-flow.split*)

lemma *sources-aux-sources*:

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

by (*induction cs rule: rev-induct, auto split: com-flow.split*)

lemma *sources-aux-sources-out*:

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

by (*induction cs rule: rev-induct, auto split: com-flow.split*)

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

$\forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \implies X \subseteq \text{sources-aux } [\langle X \rangle] \text{ vs } s \text{ f } x$

by (*subst append-Nil [symmetric], subst sources-aux.simps, auto*)

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

$\llbracket \forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \implies X \subseteq \text{sources-aux } (\langle X \rangle \# xs) \text{ vs } s \text{ f } x;$

$\forall y \in X. s: \text{dom } y \rightsquigarrow \text{dom } x \rrbracket \implies$

$X \subseteq \text{sources-aux } (\langle X \rangle \# xs @ [x']) \text{ vs } s \text{ f } x$

by (*subst append-Cons [symmetric], subst sources-aux.simps, auto split: com-flow.split*)

lemma *sources-aux-observe-hd*:

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

by (*induction cs rule: rev-induct, erule sources-aux-observe-hd-1, rule sources-aux-observe-hd-2*)

lemma *sources-aux-bval*:

assumes

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

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

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

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

proof –

have $\neg s = t (\subseteq \text{bvars } b)$

using *A and C by (erule-tac contrapos-nn, auto dest: bvars-bval)*

hence $\forall x \in S. \neg \text{bvars } b \subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# \text{cs}) \text{ vs } s \text{ f } x$
 using *A* by *blast*
 hence *D*: $\{s\}$: $\text{bvars } b \rightsquigarrow | S$
 by (*fastforce dest: sources-aux-observe-hd*)
 {
 fix *r y*
 assume $r \in A$ and $y \in S$
 moreover assume $s = r (\subseteq \text{state} \cap X)$ and $\text{state} \subseteq X$
 hence $\text{interf } s = \text{interf } r$
 by (*blast intro: interf-state*)
 ultimately have *A*: $\text{bvars } b \rightsquigarrow | \{y\}$
 using *D* by *fastforce*
 }
 with *B* and *D* show *?thesis*
 by (*fastforce simp: univ-states-if-def*)
 qed

lemma *ok-flow-aux-degen*:

assumes *A*: $\nexists S. S \neq \{\} \wedge S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$
 shows $\forall c_2' t_2 \text{vs}_2' \text{ws}_2'$.
 ok-flow-aux-1 $c_1 c_2 c_2' s_1 t_1 t_2 \text{f f}' \text{vs}_1 \text{vs}_1' \text{vs}_2 \text{vs}_2' \text{ws}_1' \text{ws}_2' \text{cs} \wedge$
 ok-flow-aux-2 $s_1 s_2 t_1 t_2 \text{f f}' \text{vs}_1 \text{vs}_1' \text{cs} \wedge$
 ok-flow-aux-3 $s_1 t_1 \text{f f}' \text{vs}_1 \text{vs}_1' \text{ws}_1 \text{ws}_1' \text{ws}_2 \text{ws}_2' \text{cs}$
 (is $\forall c_2' t_2 \text{vs}_2' \text{ws}_2'. ?P1 c_2' t_2 \text{vs}_2' \text{ws}_2' \wedge ?P2 t_2 \wedge ?P3 \text{ws}_2'$)

proof *clarify*

fix $c_2' t_2 \text{vs}_2' \text{ws}_2'$
 {
 fix *S*
 assume $S \neq \{\}$ and $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$
 hence *?P1* $c_2' t_2 \text{vs}_2' \text{ws}_2'$
 using *A* by *blast*
 }
 moreover {
 fix *S*
 assume $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$
 moreover have $\forall x. \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x \subseteq \text{sources } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x$
 by (*blast intro: subsetD [OF sources-aux-sources]*)
 ultimately have $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$
 by *blast*
 moreover assume $S \neq \{\}$
 ultimately have *?P2* t_2
 using *A* by *blast*
 }
 moreover {
 fix *S*
 assume $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$
 moreover have $\forall x. \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x \subseteq \text{sources-out } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x$
 by (*blast intro: subsetD [OF sources-aux-sources-out]*)
 ultimately have $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } \text{cs } \text{vs}_1 \text{ } s_1 \text{ f } x)\}$

by *blast*
 moreover assume $S \neq \{\}$
 ultimately have $?P3\ ws_2'$
 using A by *blast*
 }
 ultimately show $?P1\ c_2'\ t_2\ vs_2'\ ws_2' \wedge ?P2\ t_2 \wedge ?P3\ ws_2'$
 by *auto*
 qed

lemma *tags-aux-append*:
 $tags\ aux\ cs\ vs\ s\ f\ x \subseteq tags\ aux\ (cs\ @\ cs')\ vs\ s\ f\ x$
 by (*induction* cs' rule: *rev-induct*, *simp*, *subst append-assoc* [*symmetric*],
auto simp del: append-assoc split: com-flow.split)

lemma *tags-out-append*:
 $tags\ out\ cs\ vs\ s\ f\ x \subseteq tags\ out\ (cs\ @\ cs')\ vs\ s\ f\ x$
 by (*induction* cs' rule: *rev-induct*, *simp*, *subst append-assoc* [*symmetric*],
auto simp del: append-assoc split: com-flow.split)

lemma *tags-aux-tags*:
 $tags\ aux\ cs\ vs\ s\ f\ x \subseteq tags\ cs\ vs\ s\ f\ x$
 by (*induction* cs rule: *rev-induct*, *auto split: com-flow.split*)

lemma *tags-aux-tags-out*:
 $tags\ aux\ cs\ vs\ s\ f\ x \subseteq tags\ out\ cs\ vs\ s\ f\ x$
 by (*induction* cs rule: *rev-induct*, *auto split: com-flow.split*)

lemma *tags-ubound-1*:
 assumes
 $A: (y, Suc\ (length\ [c \leftarrow cs.\ c = IN\ y] + n)) \in tags\ aux\ cs\ vs\ s\ f\ x$ and
 $B: \bigwedge z\ n.\ y = z \implies$
 $(z, length\ [c \leftarrow cs.\ c = IN\ z] + n) \notin tags\ aux\ cs\ vs\ s\ f\ x$
 shows *False*
 proof –
 have $(y, length\ [c \leftarrow cs.\ c = IN\ y] + Suc\ n) \notin tags\ aux\ cs\ vs\ s\ f\ x$
 using B by *blast*
 thus *?thesis*
 using A by *simp*
 qed

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

proof –
have $(y, \text{length } [c \leftarrow cs. c = IN y] + \text{Suc } n) \notin \text{tags } cs \text{ vs } s f x$
using B **and** C **by** blast
thus $?thesis$
using A **by** simp
qed

lemma tags-ubound :
 $(y, \text{length } [c \leftarrow cs. c = IN y] + n) \notin \text{tags } cs \text{ vs } s f x$
and tags-aux-ubound :
 $(y, \text{length } [c \leftarrow cs. c = IN y] + n) \notin \text{tags-aux } cs \text{ vs } s f x$
by $(\text{induction } cs \text{ vs } s f x \text{ and } cs \text{ vs } s f x \text{ arbitrary: } n \text{ and } n$
 $\text{rule: tags-induct, auto intro: tags-ubound-1 tags-ubound-2}$
 $\text{split: if-split-asm com-flow.split-asm})$

lemma tags-out-ubound-1 :
assumes
 $A: (y, \text{Suc } (\text{length } [c \leftarrow cs. c = IN y] + n)) \in \text{tags-out } cs \text{ vs } s f x$ **and**
 $B: \bigwedge z n. y = z \implies$
 $(z, \text{length } [c \leftarrow cs. c = IN z] + n) \notin \text{tags-out } cs \text{ vs } s f x$
shows False
proof –
have $(y, \text{length } [c \leftarrow cs. c = IN y] + \text{Suc } n) \notin \text{tags-out } cs \text{ vs } s f x$
using B **by** blast
thus $?thesis$
using A **by** simp
qed

lemma tags-out-ubound :
 $(y, \text{length } [c \leftarrow cs. c = IN y] + n) \notin \text{tags-out } cs \text{ vs } s f x$
by $(\text{induction } cs \text{ vs } s f x \text{ arbitrary: } n \text{ rule: tags-out.induct, auto}$
 $\text{intro: notE [OF tags-ubound] tags-out-ubound-1}$
 $\text{split: if-split-asm com-flow.split-asm})$

lemma tags-less :
 $(y, n) \in \text{tags } cs \text{ vs } s f x \implies n < \text{length } [c \leftarrow cs. c = IN y]$
apply $(\text{rule } c\text{contr})$
apply $(\text{drule } \text{add-diff-inverse-nat})$
apply $(\text{drule } \text{ssubst, assumption})$
by $(\text{simp add: tags-ubound})$

lemma tags-aux-less :
 $(y, n) \in \text{tags-aux } cs \text{ vs } s f x \implies n < \text{length } [c \leftarrow cs. c = IN y]$
apply $(\text{rule } c\text{contr})$
apply $(\text{drule } \text{add-diff-inverse-nat})$
apply $(\text{drule } \text{ssubst, assumption})$
by $(\text{simp add: tags-aux-ubound})$

lemma *tags-out-less*:

$(y, n) \in \text{tags-out } cs \text{ vs } s f x \implies n < \text{length } [c \leftarrow cs. c = IN y]$
apply (*rule ccontr*)
apply (*drule add-diff-inverse-nat*)
apply (*drule ssubst, assumption*)
by (*simp add: tags-out-ubound*)

lemma *sources-observe-tl-1*:

assumes

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

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

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

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

E: $\bigwedge z. c = (IN z :: \text{com-flow}) \implies z \neq x \implies$
 $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$ **and**

F: $\bigwedge z. c = (OUT z :: \text{com-flow}) \implies$
 $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$ **and**

G: $\bigwedge Y b w. c = \langle Y \rangle \implies$
 $\text{sources } cs \text{ vs } s f w \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f w$

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

apply (*subst sources.simps*)

apply (*split com-flow.split*)

apply (*rule conjI*)

subgoal

proof –

show $\forall z a. c = z ::= a \longrightarrow (\text{if } z = x$
 $\text{then } \text{sources-aux } cs \text{ vs } s f x \cup \bigcup \{ \text{sources } cs \text{ vs } s f y \mid y.$
 $\text{run-flow } cs \text{ vs } s f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in \text{avars } a \}$
 $\text{else } \text{sources } cs \text{ vs } s f x) \subseteq ?F c$
(is $\forall - a. - \longrightarrow (\text{if - then } ?A \cup ?G a \text{ else } ?B) \subseteq -)$

proof (*clarify, split if-split-asm*)

fix $y z a$

assume $H: c = z ::= a$ **and** $I: z = x$

hence $?F (z ::= a) = \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x \cup$

$\bigcup \{ \text{sources } (\langle X \rangle \# cs) \text{ vs } s f y \mid y.$

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

(is - = ?A' \cup ?G' a)

by (*simp only: append-Cons [symmetric] sources.simps,*
simp add: run-flow-observe)

moreover assume $y \in ?A \cup ?G a$

moreover {

assume $y \in ?A$

hence $y \in ?A'$
 using A and H and I by *blast*
 }
 moreover {
 assume $y \in ?G a$
 hence $y \in ?G' a$
 using B and H and I by *blast*
 }
 ultimately show $y \in ?F (z ::= a)$
 by *blast*
 next
 fix $y z a$
 assume $c = z ::= a$ and $z \neq x$
 moreover from *this* have $?F (z ::= a) = \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$
 by (*simp only: append-Cons [symmetric] sources.simps, simp*)
 moreover assume $y \in ?B$
 ultimately show $y \in ?F (z ::= a)$
 using C by *blast*
 qed
 qed
 apply (*rule conjI*)
 subgoal
 proof –
 show $\forall z. c = IN z \longrightarrow (\text{if } z = x$
 then $\text{sources-aux } cs \text{ vs } s f x$ else $\text{sources } cs \text{ vs } s f x) \subseteq ?F c$
 (is $\forall -. - \longrightarrow (\text{if - then } ?A \text{ else } ?B) \subseteq -$)
 proof (*clarify, split if-split-asm*)
 fix $y z$
 assume $c = IN z$ and $z = x$
 moreover from *this* have $?F (IN z) = \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$
 by (*simp only: append-Cons [symmetric] sources.simps, simp*)
 moreover assume $y \in ?A$
 ultimately show $y \in ?F (IN z)$
 using D by *blast*
 next
 fix $y z$
 assume $c = IN z$ and $z \neq x$
 moreover from *this* have $?F (IN z) = \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$
 by (*simp only: append-Cons [symmetric] sources.simps, simp*)
 moreover assume $y \in ?B$
 ultimately show $y \in ?F (IN z)$
 using E by *blast*
 qed
 qed
 apply (*rule conjI*)
 subgoal by (*simp only: append-Cons [symmetric] sources.simps, simp add: F*)
 subgoal
 proof –
 show $\forall Y. c = \langle Y \rangle \longrightarrow \text{sources } cs \text{ vs } s f x \cup$

$\bigcup \{sources\ cs\ vs\ s\ f\ y\ |\ y.\$
 $\text{run-flow}\ cs\ vs\ s\ f:\ dom\ y\ \rightsquigarrow\ dom\ x\ \wedge\ y\ \in\ Y\} \subseteq\ ?F\ c$
 $(is\ \forall\ Y.\ -\ \longrightarrow\ ?A\ \cup\ ?G\ Y\ \subseteq\ -)$
proof clarify
fix $y\ Y$
assume $H:\ c = \langle Y \rangle$
hence $?F\ (\langle Y \rangle) = sources\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ x\ \cup$
 $\bigcup \{sources\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ y\ |\ y.\$
 $\text{run-flow}\ cs\ vs\ s\ f:\ dom\ y\ \rightsquigarrow\ dom\ x\ \wedge\ y\ \in\ Y\}$
 $(is\ - = ?A' \cup ?G' Y)$
by (*simp only: append-Cons [symmetric] sources.simps,*
simp add: run-flow-observe)
moreover assume $y \in ?A \cup ?G\ Y$
moreover {
assume $y \in ?A$
hence $y \in ?A'$
using G and H by *blast*
}
moreover {
assume $y \in ?G\ Y$
hence $y \in ?G' Y$
using G and H by *blast*
}
ultimately show $y \in ?F\ (\langle Y \rangle)$
by *blast*
qed
qed
done

lemma *sources-observe-tl-2:*

assumes
 $A:\ \bigwedge z\ a.\ c = (z ::= a :: com-flow) \implies$
 $sources\ aux\ cs\ vs\ s\ f\ x \subseteq sources\ aux\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ x$ **and**
 $B:\ \bigwedge z.\ c = (IN\ z :: com-flow) \implies$
 $sources\ aux\ cs\ vs\ s\ f\ x \subseteq sources\ aux\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ x$ **and**
 $C:\ \bigwedge z.\ c = (OUT\ z :: com-flow) \implies$
 $sources\ aux\ cs\ vs\ s\ f\ x \subseteq sources\ aux\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ x$ **and**
 $D:\ \bigwedge Y.\ c = \langle Y \rangle \implies$
 $sources\ aux\ cs\ vs\ s\ f\ x \subseteq sources\ aux\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ x$ **and**
 $E:\ \bigwedge Y\ b\ w.\ c = \langle Y \rangle \implies$
 $sources\ cs\ vs\ s\ f\ w \subseteq sources\ (\langle X \rangle\ \#\ cs)\ vs\ s\ f\ w$
shows $sources\ aux\ (cs\ @\ [c])\ vs\ s\ f\ x \subseteq$
 $sources\ aux\ (\langle X \rangle\ \#\ cs\ @\ [c])\ vs\ s\ f\ x$
 $(is\ - \subseteq\ ?F\ c)$
apply (*subst sources-aux.simps*)
apply (*split com-flow.split*)
apply (*rule conjI*)
defer
apply (*rule conjI*)

```

defer
apply (rule conjI)
defer
subgoal
proof -
  show  $\forall Y. c = \langle Y \rangle \longrightarrow \text{sources-aux } cs \text{ vs } s f x \cup$ 
     $\cup \{ \text{sources } cs \text{ vs } s f y \mid y. \text{run-flow } cs \text{ vs } s f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in Y \} \subseteq ?F c$ 
    (is  $\forall Y. - \longrightarrow ?A \cup ?G Y \subseteq -$ )
  proof clarify
    fix  $y Y$ 
    assume  $F: c = \langle Y \rangle$ 
    hence  $?F (\langle Y \rangle) = \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x \cup$ 
       $\cup \{ \text{sources } (\langle X \rangle \# cs) \text{ vs } s f y \mid y. \text{run-flow } cs \text{ vs } s f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in Y \}$ 
      (is  $- = ?A' \cup ?G' Y$ )
    by (simp only: append-Cons [symmetric] sources-aux.simps,
      simp add: run-flow-observe)
    moreover assume  $y \in ?A \cup ?G Y$ 
    moreover {
      assume  $y \in ?A$ 
      hence  $y \in ?A'$ 
      using  $D$  and  $F$  by blast
    }
    moreover {
      assume  $y \in ?G Y$ 
      hence  $y \in ?G' Y$ 
      using  $E$  and  $F$  by blast
    }
    ultimately show  $y \in ?F (\langle Y \rangle)$ 
    by blast
  qed
qed
by (simp only: append-Cons [symmetric] sources-aux.simps, simp add: A B C)+

```

```

lemma sources-observe-tl:
   $\text{sources } cs \text{ vs } s f x \subseteq \text{sources } (\langle X \rangle \# cs) \text{ vs } s f x$ 
and sources-aux-observe-tl:
   $\text{sources-aux } cs \text{ vs } s f x \subseteq \text{sources-aux } (\langle X \rangle \# cs) \text{ vs } s f x$ 
  apply (induction cs vs s f x and cs vs s f x rule: sources-induct)
  subgoal by (erule sources-observe-tl-1, assumption+)
  subgoal by (simp, subst append-Nil [symmetric], subst sources.simps, simp)
  subgoal by (erule sources-observe-tl-2, assumption+)
  by simp

```

```

lemma sources-out-observe-tl-1:
assumes
   $A: \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$ 

```

$sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$ **and**
 $B: \bigwedge z. c = (IN\ z :: com\text{-}flow) \implies$
 $sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$ **and**
 $C: \bigwedge z. c = (OUT\ z :: com\text{-}flow) \implies$
 $sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$ **and**
 $D: \bigwedge Y. c = \langle Y \rangle \implies$
 $sources\text{-}out\ cs\ vs\ s\ f\ x \subseteq sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
shows $sources\text{-}out\ (cs\ @\ [c])\ vs\ s\ f\ x \subseteq$
 $sources\text{-}out\ (\langle X \rangle \# cs\ @\ [c])\ vs\ s\ f\ x$
(is - \subseteq ?F c)
apply (*subst sources-out.simps*)
apply (*split com-flow.split*)
apply (*rule conjI*)
defer
apply (*rule conjI*)
defer
subgoal
proof
show $\forall z. c = OUT\ z \longrightarrow sources\text{-}out\ cs\ vs\ s\ f\ x \cup$
 $(if\ z = x\ then\ sources\ cs\ vs\ s\ f\ x\ else\ \{\}) \subseteq ?F\ c$
(is $\forall -. - \longrightarrow ?A \cup (if\ -\ then\ ?B\ else\ -) \subseteq -$)
proof (*clarify, split if-split-asm*)
fix $y\ z$
assume $E: c = OUT\ z$ **and** $F: z = x$
assume $y \in ?A \cup ?B$
moreover {
assume $y \in ?A$
hence $y \in sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
using C **and** E **by** *blast*
}
moreover {
assume $y \in ?B$
hence $y \in sources\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
by (*rule subsetD [OF sources-observe-tl]*)
}
ultimately show $y \in ?F\ (OUT\ z)$
using F **by** (*simp only: append-Cons [symmetric] sources-out.simps,*
auto)
next
fix $y\ z$
assume $c = OUT\ z$ **and** $y \in sources\text{-}out\ cs\ vs\ s\ f\ x \cup \{\}$
hence $y \in sources\text{-}out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
using C **by** *blast*
thus $y \in ?F\ (OUT\ z)$
by (*simp only: append-Cons [symmetric] sources-out.simps, simp*)
qed
next
show $\forall Y. c = \langle Y \rangle \longrightarrow sources\text{-}out\ cs\ vs\ s\ f\ x \cup$
 $\bigcup\ \{sources\ cs\ vs\ s\ f\ y\ \mid\ y.\}$

$run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in Y \} \subseteq ?F\ c$
 $(is\ \forall Y. - \longrightarrow ?A \cup ?G\ Y \subseteq -)$
proof *clarify*
fix $y\ Y$
assume $E: c = \langle Y \rangle$
assume $y \in ?A \cup ?G\ Y$
moreover {
assume $y \in ?A$
hence $y \in sources-out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
using D and E **by** *blast*
}
moreover {
assume $y \in ?G\ Y$
hence $y \in \bigcup \{sources\ (\langle X \rangle \# cs)\ vs\ s\ f\ y \mid y.\$
 $run-flow\ (\langle X \rangle \# cs)\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in Y\}$
by (*auto intro: subsetD [OF sources-observe-tl]*)
simp: run-flow-observe)
}
ultimately show $y \in ?F\ (\langle Y \rangle)$
by (*simp only: append-Cons [symmetric] sources-out.simps, auto*)
qed
qed
by (*simp only: append-Cons [symmetric] sources-out.simps, simp add: A B*)+

lemma *sources-out-observe-tl:*
 $sources-out\ cs\ vs\ s\ f\ x \subseteq sources-out\ (\langle X \rangle \# cs)\ vs\ s\ f\ x$
by (*induction cs vs s f x rule: sources-out.induct,*
erule sources-out-observe-tl-1, simp-all)

lemma *tags-observe-tl-1:*
 $\llbracket \bigwedge z\ a. c = z ::= a \implies z = x \implies$
 $tags-aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags-aux\ cs\ vs\ s\ f\ x;$
 $\bigwedge z\ a\ b\ w. c = z ::= a \implies z = x \implies$
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ w = tags\ cs\ vs\ s\ f\ w;$
 $\bigwedge z\ a. c = z ::= a \implies z \neq x \implies$
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$
 $\bigwedge z. c = IN\ z \implies z = x \implies$
 $tags-aux\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags-aux\ cs\ vs\ s\ f\ x;$
 $\bigwedge z. c = IN\ z \implies z \neq x \implies$
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$
 $\bigwedge z. c = OUT\ z \implies$
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ x = tags\ cs\ vs\ s\ f\ x;$
 $\bigwedge Y\ b\ w. c = \langle Y \rangle \implies$
 $tags\ (\langle X \rangle \# cs)\ vs\ s\ f\ w = tags\ cs\ vs\ s\ f\ w \rrbracket \implies$
 $tags\ (\langle X \rangle \# cs\ @\ [c])\ vs\ s\ f\ x = tags\ (cs\ @\ [c])\ vs\ s\ f\ x$
by (*subst tags.simps, split com-flow.split, simp-all only: append-Cons*
[symmetric] tags.simps, simp-all add: run-flow-observe)

lemma *tags-observe-tl-2*:

$\llbracket \bigwedge z a. c = z ::= a \implies$
 $\quad \text{tags-aux } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-aux } cs \text{ vs } s f x;$
 $\bigwedge z. c = IN z \implies$
 $\quad \text{tags-aux } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-aux } cs \text{ vs } s f x;$
 $\bigwedge z. c = OUT z \implies$
 $\quad \text{tags-aux } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-aux } cs \text{ vs } s f x;$
 $\bigwedge Y. c = \langle Y \rangle \implies$
 $\quad \text{tags-aux } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-aux } cs \text{ vs } s f x;$
 $\bigwedge Y b w. c = \langle Y \rangle \implies$
 $\quad \text{tags } (\langle X \rangle \# cs) \text{ vs } s f w = \text{tags } cs \text{ vs } s f w \rrbracket \implies$
 $\quad \text{tags-aux } (\langle X \rangle \# cs @ [c]) \text{ vs } s f x = \text{tags-aux } (cs @ [c]) \text{ vs } s f x$
by (*subst tags-aux.simps, split com-flow.split, simp-all only: append-Cons*
[symmetric] tags-aux.simps, simp-all add: run-flow-observe)

lemma *tags-observe-tl*:

$\text{tags } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags } cs \text{ vs } s f x$
and *tags-aux-observe-tl*:
 $\text{tags-aux } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-aux } cs \text{ vs } s f x$
apply (*induction cs vs s f x and cs vs s f x rule: tags-induct*)
subgoal by (*erule tags-observe-tl-1, assumption+*)
subgoal by (*subst append-Nil [symmetric], subst tags.simps tags-aux.simps, simp*)
subgoal by (*erule tags-observe-tl-2, assumption+*)
subgoal by (*subst append-Nil [symmetric], subst tags.simps tags-aux.simps, simp*)
done

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

$\llbracket \bigwedge z a. c = z ::= a \implies$
 $\quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x;$
 $\bigwedge z. c = IN z \implies$
 $\quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x;$
 $\bigwedge z. c = OUT z \implies$
 $\quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x;$
 $\bigwedge Y. c = \langle Y \rangle \implies$
 $\quad \text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x \rrbracket \implies$
 $\quad \text{tags-out } (\langle X \rangle \# cs @ [c]) \text{ vs } s f x = \text{tags-out } (cs @ [c]) \text{ vs } s f x$
by (*subst tags-out.simps, split com-flow.split, simp-all only: append-Cons*
[symmetric] tags-out.simps, simp-all add: run-flow-observe tags-observe-tl)

lemma *tags-out-observe-tl*:

$\text{tags-out } (\langle X \rangle \# cs) \text{ vs } s f x = \text{tags-out } cs \text{ vs } s f x$
apply (*induction cs vs s f x rule: tags-out.induct*)
apply (*erule tags-out-observe-tl-1, assumption+*)
by (*subst append-Nil [symmetric], subst tags-out.simps, simp*)

lemma *tags-sources-1*:

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

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

assumes

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

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

proof –

have $I: n < \text{length } [c \leftarrow cs @ [c]. c = \text{IN } y]$
using H **by** (rule tags-less)
hence $?F (cs @ [c]) = ?F cs$
using le-Suc-eq **by** auto
moreover have $c \neq \text{IN } y \vee n < \text{length } [c \leftarrow cs. c = \text{IN } y] \implies$
 $\text{Suc } (\text{Max } (?F \text{ cs})) \leq \text{length } cs$
 $(\text{is } - \implies ?m \leq -)$
using I **by** $(\text{subst Suc-le-eq, subst Max-less-iff,}$
 $\text{auto elim: le-neq-implies-less})$
ultimately have $J: c \neq \text{IN } y \vee n < \text{length } [c \leftarrow cs. c = \text{IN } y] \implies$
 $\text{take } (\text{Suc } (\text{Max } (?F (cs @ [c])))) (cs @ [c]) = \text{take } ?m \text{ cs } \wedge$
 $\text{drop } (\text{Suc } (\text{Max } (?F (cs @ [c])))) (cs @ [c]) = \text{drop } ?m \text{ cs } @ [c]$
by simp

from H **show** $?thesis$

proof $(\text{subst } (asm) \text{ tags.simps, split com-flow.split-asm})$

fix $z a$

assume $K: c = z ::= a$

show $(y, n) \in (\text{if } z = x$

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

$run-flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in avars\ a\}$
 $else\ tags\ cs\ vs\ s\ f\ x) \implies ?thesis$
 $(is\ - \in (if\ -\ then\ ?A \cup ?B\ else\ ?C) \implies -)$
proof (*split if-split-asm*)
assume $L: z = x$ **and** $(y, n) \in ?A \cup ?B$
moreover {
assume $(y, n) \in ?A$
hence $y \in sources\ aux\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ x$
using A **and** K **and** L **by** *simp*
}
moreover {
assume $(y, n) \in ?B$
hence $y \in \bigcup \{sources\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ y \mid y.\$
 $run-flow\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f:$
 $dom\ y \rightsquigarrow dom\ x \wedge y \in avars\ a\}$
using B **and** K **and** L **by** (*auto simp: run-flow-append [symmetric]*)
}
ultimately show *?thesis*
using J **and** K **by** *auto*
next
assume $z \neq x$ **and** $(y, n) \in ?C$
moreover from this have
 $y \in sources\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ x$
using C **and** K **by** *simp*
ultimately show *?thesis*
using J **and** K **by** *simp*
qed
next
fix z
assume $K: c = IN\ z$
show $(y, n) \in (if\ z = x$
 $then\ insert\ (x,\ length\ [c \leftarrow cs.\ c = IN\ x])\ (tags\ aux\ cs\ vs\ s\ f\ x)$
 $else\ tags\ cs\ vs\ s\ f\ x) \implies ?thesis$
 $(is\ - \in (if\ -\ then\ insert\ -\ ?A\ else\ ?B) \implies -)$
proof (*split if-split-asm, erule insertE*)
assume $(y, n) = (x,\ length\ [c \leftarrow cs.\ c = IN\ x])$ **and** $z = x$
moreover from this have $Max\ (?F\ (cs\ @\ [c])) = length\ cs$
using K **by** (*subst Max-eq-iff, auto elim: le-SucE*)
ultimately show *?thesis*
by *simp*
next
assume $L: (y, n) \in tags\ aux\ cs\ vs\ s\ f\ x$ **and** $z = x$
moreover from this have
 $y \in sources\ aux\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ x$
using D **and** K **by** *simp*
moreover have $n < length\ [c \leftarrow cs.\ c = IN\ y]$
using L **by** (*rule tags-aux-less*)
ultimately show *?thesis*
using J **and** K **by** *simp*

```

next
  assume  $L: (y, n) \in \text{tags } cs \text{ vs } s f x$  and  $z \neq x$ 
  moreover from this have
     $y \in \text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$ 
    using  $E$  and  $K$  by simp
  moreover have  $n < \text{length } [c \leftarrow cs. c = IN y]$ 
    using  $L$  by (rule tags-less)
  ultimately show ?thesis
    using  $J$  and  $K$  by simp
qed
next
fix  $z$ 
assume  $c = OUT z$  and  $(y, n) \in \text{tags } cs \text{ vs } s f x$ 
moreover from this have
   $y \in \text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$ 
  using  $F$  by simp
ultimately show ?thesis
  using  $J$  by simp
next
fix  $X$ 
assume  $K: c = \langle X \rangle$ 
assume  $(y, n) \in \text{tags } cs \text{ vs } s f x \cup \bigcup \{ \text{tags } cs \text{ vs } s f y \mid y. \text{run-flow } cs \text{ vs } s f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X \}$ 
  (is -  $\in ?A \cup ?B$ )
moreover {
  assume  $(y, n) \in ?A$ 
  hence  $y \in \text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$ 
    using  $G$  and  $K$  by simp
}
moreover {
  assume  $(y, n) \in ?B$ 
  hence  $y \in \bigcup \{ \text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f y \mid y. \text{run-flow } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X \}$ 
    using  $G$  and  $K$  by (auto simp: run-flow-append [symmetric])
}
ultimately show ?thesis
  using  $J$  and  $K$  by auto
qed
qed

```

lemma *tags-sources-2*:

```

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

```

(is $\bigwedge - . - \implies - \implies$ let $m = \text{Suc} (\text{Max} (?F \text{ cs}))$ in
 $- \in \text{sources-aux} - (?G \text{ m cs}) (?H \text{ m cs}) -$)

assumes

$B: \bigwedge z. c = (\text{IN } z :: \text{com-flow}) \implies$

$(y, n) \in \text{tags-aux cs vs s f x} \implies$ let $m = \text{Suc} (\text{Max} (?F \text{ cs}))$ in
 $y \in \text{sources-aux} (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) \text{ f x}$ **and**

$C: \bigwedge z. c = (\text{OUT } z :: \text{com-flow}) \implies$

$(y, n) \in \text{tags-aux cs vs s f x} \implies$ let $m = \text{Suc} (\text{Max} (?F \text{ cs}))$ in
 $y \in \text{sources-aux} (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) \text{ f x}$ **and**

$D: \bigwedge X. c = \langle X \rangle \implies$

$(y, n) \in \text{tags-aux cs vs s f x} \implies$ let $m = \text{Suc} (\text{Max} (?F \text{ cs}))$ in
 $y \in \text{sources-aux} (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) \text{ f x}$ **and**

$E: \bigwedge X \text{ b w. } c = \langle X \rangle \implies$

$(y, n) \in \text{tags cs vs s f w} \implies$ let $m = \text{Suc} (\text{Max} (?F \text{ cs}))$ in
 $y \in \text{sources} (\text{drop } m \text{ cs}) (?G \text{ m cs}) (?H \text{ m cs}) \text{ f w}$ **and**

$F: (y, n) \in \text{tags-aux} (\text{cs } @ [c]) \text{ vs s f x}$

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

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

proof –

have $G: n < \text{length} [c \leftarrow \text{cs } @ [c]. c = \text{IN } y]$

using F **by** (rule tags-aux-less)

hence $?F (\text{cs } @ [c]) = ?F \text{ cs}$

using le-Suc-eq **by** auto

moreover have $c \neq \text{IN } y \vee n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y] \implies$

$\text{Suc} (\text{Max} (?F \text{ cs})) \leq \text{length } \text{cs}$

(is $- \implies ?m \leq -$)

using G **by** (subst Suc-le-eq, subst Max-less-iff,

auto elim: le-neq-implies-less)

ultimately have $H: c \neq \text{IN } y \vee n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y] \implies$

$\text{take} (\text{Suc} (\text{Max} (?F (\text{cs } @ [c])))) (\text{cs } @ [c]) = \text{take } ?m \text{ cs} \wedge$

$\text{drop} (\text{Suc} (\text{Max} (?F (\text{cs } @ [c])))) (\text{cs } @ [c]) = \text{drop } ?m \text{ cs } @ [c]$

by simp

from F **show** ?thesis

proof (subst (asm) tags-aux.simps, split com-flow.split-asm)

fix $z \ a$

assume $c = z ::= a$ **and** $(y, n) \in \text{tags-aux cs vs s f x}$

moreover from this have

$y \in \text{sources-aux} (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) \text{ f x}$

using A **by** simp

ultimately show ?thesis

using H **by** simp

next

fix z

assume $c = \text{IN } z$ **and** $I: (y, n) \in \text{tags-aux cs vs s f x}$

moreover from this have

$y \in \text{sources-aux} (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) \text{ f x}$

using B **by** simp

moreover have $n < \text{length} [c \leftarrow \text{cs}. c = \text{IN } y]$

using I **by** (rule tags-aux-less)

ultimately show $?thesis$
using H **by** *simp*
next
fix z
assume $c = OUT\ z$ **and** $(y, n) \in tags\ aux\ cs\ vs\ s\ f\ x$
moreover from this have
 $y \in sources\ aux\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ x$
using C **by** *simp*
ultimately show $?thesis$
using H **by** *simp*
next
fix X
assume $I: c = \langle X \rangle$
assume $(y, n) \in tags\ aux\ cs\ vs\ s\ f\ x \cup \bigcup \{tags\ cs\ vs\ s\ f\ y \mid y.$
 $run\ flow\ cs\ vs\ s\ f: dom\ y \rightsquigarrow dom\ x \wedge y \in X\}$
 $(is\ - \in ?A \cup ?B)$
moreover {
assume $(y, n) \in ?A$
hence $y \in sources\ aux\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ x$
using D **and** I **by** *simp*
}
moreover {
assume $(y, n) \in ?B$
hence $y \in \bigcup \{sources\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f\ y \mid y.$
 $run\ flow\ (drop\ ?m\ cs)\ (?G\ ?m\ cs)\ (?H\ ?m\ cs)\ f:$
 $dom\ y \rightsquigarrow dom\ x \wedge y \in X\}$
using E **and** I **by** (*auto simp: run-flow-append [symmetric]*)
}
ultimately show $?thesis$
using H **and** I **by** *auto*
qed
qed

lemma *tags-sources:*

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

and *tags-aux-sources-aux:*

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

by (*induction cs vs s f x and cs vs s f x rule: tags-induct,*
erule-tac [3] tags-sources-2, erule tags-sources-1, simp-all)

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

assumes

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

assumes

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

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

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

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

proof –

have $F: n < \text{length } [c \leftarrow cs @ [c]. c = \text{IN } y]$

using E **by** (*rule tags-out-less*)

hence $?F (cs @ [c]) = ?F cs$

using *le-Suc-eq* **by** *auto*

moreover have $c \neq \text{IN } y \vee n < \text{length } [c \leftarrow cs. c = \text{IN } y] \implies$

$\text{Suc } (\text{Max } (?F cs)) \leq \text{length } cs$

$(\text{is } - \implies ?m \leq -)$

using F **by** (*subst Suc-le-eq, subst Max-less-iff,*

auto elim: le-neq-implies-less)

ultimately have $G: c \neq \text{IN } y \vee n < \text{length } [c \leftarrow cs. c = \text{IN } y] \implies$

$\text{take } (\text{Suc } (\text{Max } (?F (cs @ [c])))) (cs @ [c]) = \text{take } ?m \text{ cs} \wedge$

$\text{drop } (\text{Suc } (\text{Max } (?F (cs @ [c])))) (cs @ [c]) = \text{drop } ?m \text{ cs} @ [c]$

by *simp*

from E **show** *?thesis*

proof (*subst (asm) tags-out.simps, split com-flow.split-asm*)

fix $z a$

assume $c = z ::= a$ **and** $(y, n) \in \text{tags-out } cs \text{ vs } s f x$

moreover from this have

$y \in \text{sources-out } (\text{drop } ?m \text{ cs}) (?G \text{ } ?m \text{ cs}) (?H \text{ } ?m \text{ cs}) f x$

using A **by** *simp*

ultimately show *?thesis*

using G **by** *simp*

next

fix z

assume $c = \text{IN } z$ **and** $H: (y, n) \in \text{tags-out } cs \text{ vs } s f x$

moreover from *this* have
 $y \in \text{sources-out } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$
using B by *simp*
moreover have $n < \text{length } [c \leftarrow \text{cs}. c = \text{IN } y]$
using H by (*rule tags-out-less*)
ultimately show *?thesis*
using G by *simp*
next
fix z
assume $H: c = \text{OUT } z$
show $(y, n) \in \text{tags-out cs vs s f x} \cup$
 $(\text{if } z = x \text{ then tags cs vs s f x else } \{\}) \implies ?thesis$
 $(\text{is } - \in ?A \cup (\text{if - then } ?B \text{ else -}) \implies -)$
proof (*split if-split-asm*)
assume $z = x$ and $(y, n) \in ?A \cup ?B$
moreover {
assume $(y, n) \in ?A$
hence $y \in \text{sources-out } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$
using C and H by *simp*
}
moreover {
assume $(y, n) \in ?B$
hence $y \in \text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$
by (*auto dest: tags-sources*)
}
ultimately show *?thesis*
using G and H by *auto*
next
assume $(y, n) \in ?A \cup \{\}$
moreover from *this* have
 $y \in \text{sources-out } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$
using C and H by *simp*
ultimately show *?thesis*
using G and H by *simp*
qed
next
fix X
assume $H: c = \langle X \rangle$
assume $(y, n) \in \text{tags-out cs vs s f x} \cup \bigcup \{\text{tags cs vs s f y} \mid y.$
 $\text{run-flow cs vs s f: dom } y \rightsquigarrow \text{dom } x \wedge y \in X\}$
 $(\text{is } - \in ?A \cup ?B)$
moreover {
assume $(y, n) \in ?A$
hence $y \in \text{sources-out } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f x$
using D and H by *simp*
}
moreover {
assume $(y, n) \in ?B$
hence $y \in \bigcup \{\text{sources } (\text{drop } ?m \text{ cs}) (?G ?m \text{ cs}) (?H ?m \text{ cs}) f y \mid y.$

$run_flow (drop \ ?m \ cs) (\ ?G \ ?m \ cs) (\ ?H \ ?m \ cs) f$:
 $dom \ y \rightsquigarrow dom \ x \wedge y \in X$
by (*fastforce dest: tags-sources simp: run-flow-append [symmetric]*)
}
ultimately show *?thesis*
using *G* and *H* **by** *auto*
qed
qed

lemma *tags-out-sources-out*:
 $(y, n) \in tags_out \ cs \ vs \ s \ f \ x \implies$
 $let \ m = Suc \ (Max \ \{k. \ k \leq length \ cs \wedge$
 $length \ [c \leftarrow take \ k \ cs. \ c = IN \ y] \leq n\})$
 $in \ y \in sources_out \ (drop \ m \ cs) \ (vs \ @ \ in_flow \ (take \ m \ cs) \ vs \ f)$
 $(run_flow \ (take \ m \ cs) \ vs \ s \ f) \ f \ x$
by (*induction cs vs s f x rule: tags-out.induct,*
erule tags-out-sources-out-1, simp-all)

lemma *sources-member-1*:

assumes

$A: \bigwedge z \ a \ b \ w. \ c = (z ::= a :: com_flow) \implies z = x \implies$
 $y \in sources_aux \ cs' \ (vs \ @ \ in_flow \ cs \ vs \ f) \ (run_flow \ cs \ vs \ s \ f) \ f \ x \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources_aux \ (cs \ @ \ cs') \ vs \ s \ f \ x$
 $(is \ \bigwedge - \ . \ - \implies - \implies - \in sources_aux \ - \ ?vs' \ ?s' \ - \ - \implies$
 $- \subseteq sources_aux \ ?cs \ - \ - \ -)$

assumes

$B: \bigwedge z \ a \ b \ w. \ c = (z ::= a :: com_flow) \implies z = x \implies$
 $y \in sources \ cs' \ ?vs' \ ?s' \ f \ w \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources \ ?cs \ vs \ s \ f \ w$ **and**
 $C: \bigwedge z \ a. \ c = (z ::= a :: com_flow) \implies z \neq x \implies$
 $y \in sources \ cs' \ ?vs' \ ?s' \ f \ x \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources \ ?cs \ vs \ s \ f \ x$ **and**
 $D: \bigwedge z. \ c = (IN \ z :: com_flow) \implies z = x \implies$
 $y \in sources_aux \ cs' \ ?vs' \ ?s' \ f \ x \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources_aux \ ?cs \ vs \ s \ f \ x$ **and**
 $E: \bigwedge z. \ c = (IN \ z :: com_flow) \implies z \neq x \implies$
 $y \in sources \ cs' \ ?vs' \ ?s' \ f \ x \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources \ ?cs \ vs \ s \ f \ x$ **and**
 $F: \bigwedge z. \ c = (OUT \ z :: com_flow) \implies$
 $y \in sources \ cs' \ ?vs' \ ?s' \ f \ x \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources \ ?cs \ vs \ s \ f \ x$ **and**
 $G: \bigwedge X \ b \ w. \ c = \langle X \rangle \implies$
 $y \in sources \ cs' \ ?vs' \ ?s' \ f \ w \implies$
 $sources \ cs \ vs \ s \ f \ y \subseteq sources \ ?cs \ vs \ s \ f \ w$

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

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

proof (*subst (asm) sources.simps, split com-flow.split-asm*)

fix *z a*

```

assume  $H: c = z ::= a$ 
show  $y \in (\text{if } z = x$ 
   $\text{then sources-aux } cs' ?vs' ?s' f x \cup \bigcup \{ \text{sources } cs' ?vs' ?s' f w \mid w.$ 
     $\text{run-flow } cs' ?vs' ?s' f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in \text{avars } a \}$ 
   $\text{else sources } cs' ?vs' ?s' f x) \implies ?thesis$ 
   $(\text{is } - \in (\text{if } - \text{ then } ?A \cup ?B \text{ else } ?C) \implies -)$ 
proof (split if-split-asm)
  assume  $I: z = x$  and  $y \in ?A \cup ?B$ 
  moreover {
    assume  $y \in ?A$ 
    hence  $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$ 
    using  $A$  and  $H$  and  $I$  by simp
  }
  moreover {
    assume  $y \in ?B$ 
    hence  $\text{sources } cs \text{ vs } s f y \subseteq \bigcup \{ \text{sources } ?cs \text{ vs } s f w \mid w.$ 
       $\text{run-flow } ?cs \text{ vs } s f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in \text{avars } a \}$ 
    using  $B$  and  $H$  and  $I$  by (fastforce simp: run-flow-append)
  }
  ultimately show  $?thesis$ 
  using  $H$  by (simp only: append-assoc [symmetric] sources.simps, auto)
next
  assume  $z \neq x$  and  $y \in ?C$ 
  moreover from this have  $\text{sources } cs \text{ vs } s f y \subseteq \text{sources } ?cs \text{ vs } s f x$ 
  using  $C$  and  $H$  by simp
  ultimately show  $?thesis$ 
  using  $H$  by (simp only: append-assoc [symmetric] sources.simps, auto)
qed
next
fix  $z$ 
assume  $H: c = IN z$ 
show  $y \in (\text{if } z = x$ 
   $\text{then sources-aux } cs' ?vs' ?s' f x$ 
   $\text{else sources } cs' ?vs' ?s' f x) \implies ?thesis$ 
   $(\text{is } - \in (\text{if } - \text{ then } ?A \text{ else } ?B) \implies -)$ 
proof (split if-split-asm)
  assume  $z = x$  and  $y \in ?A$ 
  moreover from this have  $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$ 
  using  $D$  and  $H$  by simp
  ultimately show  $?thesis$ 
  using  $H$  by (simp only: append-assoc [symmetric] sources.simps, auto)
next
  assume  $z \neq x$  and  $y \in ?B$ 
  moreover from this have  $\text{sources } cs \text{ vs } s f y \subseteq \text{sources } ?cs \text{ vs } s f x$ 
  using  $E$  and  $H$  by simp
  ultimately show  $?thesis$ 
  using  $H$  by (simp only: append-assoc [symmetric] sources.simps, auto)
qed
next

```

```

fix z
assume c = OUT z and y ∈ sources cs' ?vs' ?s' f x
moreover from this have sources cs vs s f y ⊆ sources ?cs vs s f x
  using F by simp
ultimately show ?thesis
  by (simp only: append-assoc [symmetric] sources.simps, auto)
next
fix X
assume H: c = ⟨X⟩
assume y ∈ sources cs' ?vs' ?s' f x ∪ ∪ {sources cs' ?vs' ?s' f w | w.
  run-flow cs' ?vs' ?s' f: dom w ≈ dom x ∧ w ∈ X}
  (is - ∈ ?A ∪ ?B)
moreover {
  assume y ∈ ?A
  hence sources cs vs s f y ⊆ sources ?cs vs s f x
  using G and H by simp
}
moreover {
  assume y ∈ ?B
  hence sources cs vs s f y ⊆ ∪ {sources ?cs vs s f w | w.
  run-flow ?cs vs s f: dom w ≈ dom x ∧ w ∈ X}
  using G and H by (auto simp: run-flow-append)
}
ultimately show ?thesis
using H by (simp only: append-assoc [symmetric] sources.simps, auto)
qed

```

lemma sources-member-2:

assumes

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

assumes

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

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

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

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

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

proof (*subst (asm) sources-aux.simps, split com-flow.split-asm*)
fix z a
assume $c = z ::= a$ **and** $y \in \text{sources-aux } cs' ?vs' ?s' f x$
moreover from this have $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$
using A **by** *simp*
ultimately show *?thesis*
by (*simp only: append-assoc [symmetric] sources-aux.simps, auto*)
next
fix z
assume $c = IN z$ **and** $y \in \text{sources-aux } cs' ?vs' ?s' f x$
moreover from this have $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$
using B **by** *simp*
ultimately show *?thesis*
by (*simp only: append-assoc [symmetric] sources-aux.simps, auto*)
next
fix z
assume $c = OUT z$ **and** $y \in \text{sources-aux } cs' ?vs' ?s' f x$
moreover from this have $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$
using C **by** *simp*
ultimately show *?thesis*
by (*simp only: append-assoc [symmetric] sources-aux.simps, auto*)
next
fix X
assume $F: c = \langle X \rangle$
assume $y \in \text{sources-aux } cs' ?vs' ?s' f x \cup \bigcup \{ \text{sources } cs' ?vs' ?s' f w \mid w. \text{run-flow } cs' ?vs' ?s' f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$
(is - $\in ?A \cup ?B$)
moreover {
assume $y \in ?A$
hence $\text{sources } cs \text{ vs } s f y \subseteq \text{sources-aux } ?cs \text{ vs } s f x$
using D **and** F **by** *simp*
}
moreover {
assume $y \in ?B$
hence $\text{sources } cs \text{ vs } s f y \subseteq \bigcup \{ \text{sources } ?cs \text{ vs } s f w \mid w. \text{run-flow } ?cs \text{ vs } s f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$
using E **and** F **by** (*auto simp: run-flow-append*)
}
ultimately show *?thesis*
using F **by** (*simp only: append-assoc [symmetric] sources-aux.simps, auto*)
qed

lemma *sources-member:*

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

and *sources-aux-member:*

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

by (*induction cs' vs s f x and cs' vs s f x rule: sources-induct,*

erule-tac [3] *sources-member-2*, *erule sources-member-1*, *simp-all*)

lemma *sources-out-member-1*:

assumes

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

assumes

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

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

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

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

proof (*subst (asm) sources-out.simps, split com-flow.split-asm*)

fix $z a$

assume $c = z ::= a$ **and** $y \in \text{sources-out } cs' ?vs' ?s' f x$

moreover from this have $\text{sources } cs \text{ vs } s \text{ } f y \subseteq \text{sources-out } ?cs \text{ vs } s \text{ } f x$
using A **by** *simp*

ultimately show *?thesis*

by (*simp only: append-assoc [symmetric] sources-out.simps, auto*)

next

fix z

assume $c = IN z$ **and** $y \in \text{sources-out } cs' ?vs' ?s' f x$

moreover from this have $\text{sources } cs \text{ vs } s \text{ } f y \subseteq \text{sources-out } ?cs \text{ vs } s \text{ } f x$
using B **by** *simp*

ultimately show *?thesis*

by (*simp only: append-assoc [symmetric] sources-out.simps, auto*)

next

fix z

assume $E: c = OUT z$

show $y \in \text{sources-out } cs' ?vs' ?s' f x \cup$

$(\text{if } z = x \text{ then } \text{sources } cs' ?vs' ?s' f x \text{ else } \{\}) \implies ?thesis$

$(\text{is } - \in ?A \cup (\text{if } - \text{ then } ?B \text{ else } -) \implies -)$

proof (*split if-split-asm*)

assume $z = x$ **and** $y \in ?A \cup ?B$

moreover {

assume $y \in ?A$

hence $\text{sources } cs \text{ vs } s \text{ } f y \subseteq \text{sources-out } ?cs \text{ vs } s \text{ } f x$

using C **and** E **by** *simp*

}

```

moreover {
  assume  $y \in ?B$ 
  hence  $\text{sources } cs \text{ vs } s \text{ f } y \subseteq \text{sources } ?cs \text{ vs } s \text{ f } x$ 
  by (rule sources-member)
}
ultimately show ?thesis
using  $E$  by (simp only: append-assoc [symmetric] sources-out.simps, auto)
next
assume  $y \in ?A \cup \{\}$ 
moreover from this have  $\text{sources } cs \text{ vs } s \text{ f } y \subseteq \text{sources-out } ?cs \text{ vs } s \text{ f } x$ 
using  $C$  and  $E$  by simp
ultimately show ?thesis
using  $E$  by (simp only: append-assoc [symmetric] sources-out.simps, auto)
qed
next
fix  $X$ 
assume  $E: c = \langle X \rangle$ 
assume  $y \in \text{sources-out } cs' \text{ ?vs}' \text{ ?s}' \text{ f } x \cup \bigcup \{ \text{sources } cs' \text{ ?vs}' \text{ ?s}' \text{ f } w \mid w. \text{run-flow } cs' \text{ ?vs}' \text{ ?s}' \text{ f}: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$ 
(is  $- \in ?A \cup ?B$ )
moreover {
  assume  $y \in ?A$ 
  hence  $\text{sources } cs \text{ vs } s \text{ f } y \subseteq \text{sources-out } ?cs \text{ vs } s \text{ f } x$ 
  using  $D$  and  $E$  by simp
}
moreover {
  assume  $y \in ?B$ 
  hence  $\text{sources } cs \text{ vs } s \text{ f } y \subseteq \bigcup \{ \text{sources } ?cs \text{ vs } s \text{ f } w \mid w. \text{run-flow } ?cs \text{ vs } s \text{ f}: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$ 
  by (auto dest: sources-member simp: run-flow-append)
}
ultimately show ?thesis
using  $E$  by (simp only: append-assoc [symmetric] sources-out.simps, auto)
qed

```

lemma *sources-out-member:*

```

 $y \in \text{sources-out } cs' \text{ (vs @ in-flow } cs \text{ vs } f) \text{ (run-flow } cs \text{ vs } s \text{ f) } f \text{ x} \implies$ 
 $\text{sources } cs \text{ vs } s \text{ f } y \subseteq \text{sources-out } (cs \text{ @ } cs') \text{ vs } s \text{ f } x$ 
by (induction cs' vs s f x rule: sources-out.induct,
erule sources-out-member-1, simp-all)

```

lemma *tags-member-1:*

assumes

```

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

```

assumes

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

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

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

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

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

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

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

proof (*subst (asm) sources.simps, split com-flow.split-asm*)

fix $z a$

assume $H: c = z ::= a$

show $y \in (\text{if } z = x$
 $\text{then sources-aux } cs' ?vs' ?s' f x \cup \bigcup \{\text{sources } cs' ?vs' ?s' f w \mid w.$
 $\text{run-flow } cs' ?vs' ?s' f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in \text{avars } a\}$
 $\text{else sources } cs' ?vs' ?s' f x) \implies ?thesis$
 $(\text{is } - \in (\text{if } - \text{ then } ?A \cup ?B \text{ else } ?C) \implies -)$

proof (*split if-split-asm*)

assume $I: z = x$ **and** $y \in ?A \cup ?B$

moreover {
assume $y \in ?A$
hence $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-aux } ?cs \text{ vs } s f x$
using A **and** H **and** I **by** *simp*
}

moreover {
assume $y \in ?B$
hence $\text{tags } cs \text{ vs } s f y \subseteq \bigcup \{\text{tags } ?cs \text{ vs } s f w \mid w.$
 $\text{run-flow } ?cs \text{ vs } s f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in \text{avars } a\}$
using B **and** H **and** I **by** (*fastforce simp: run-flow-append*)
}

ultimately show $?thesis$
using H **by** (*simp only: append-assoc [symmetric] tags.simps, auto*)

next

assume $z \neq x$ **and** $y \in ?C$

moreover from *this* **have** $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } ?cs \text{ vs } s f x$
using C **and** H **by** *simp*
ultimately show $?thesis$

```

    using  $H$  by (simp only: append-assoc [symmetric] tags.simps, auto)
  qed
next
  fix  $z$ 
  assume  $H: c = IN\ z$ 
  show  $y \in (if\ z = x$ 
    then sources-aux  $cs' ?vs' ?s' f\ x$ 
    else sources  $cs' ?vs' ?s' f\ x) \implies ?thesis$ 
    (is  $- \in (if\ -\ then\ ?A\ else\ ?B) \implies -$ )
  proof (split if-split-asm)
    assume  $z = x$  and  $y \in ?A$ 
    moreover from this have tags  $cs\ vs\ s\ f\ y \subseteq tags\ aux\ ?cs\ vs\ s\ f\ x$ 
      using  $D$  and  $H$  by simp
    ultimately show ?thesis
      using  $H$  by (simp only: append-assoc [symmetric] tags.simps, auto)
  next
    assume  $z \neq x$  and  $y \in ?B$ 
    moreover from this have tags  $cs\ vs\ s\ f\ y \subseteq tags\ ?cs\ vs\ s\ f\ x$ 
      using  $E$  and  $H$  by simp
    ultimately show ?thesis
      using  $H$  by (simp only: append-assoc [symmetric] tags.simps, auto)
  qed
next
  fix  $z$ 
  assume  $c = OUT\ z$  and  $y \in sources\ cs' ?vs' ?s' f\ x$ 
  moreover from this have tags  $cs\ vs\ s\ f\ y \subseteq tags\ ?cs\ vs\ s\ f\ x$ 
    using  $F$  by simp
  ultimately show ?thesis
    by (simp only: append-assoc [symmetric] tags.simps, auto)
next
  fix  $X$ 
  assume  $H: c = \langle X \rangle$ 
  assume  $y \in sources\ cs' ?vs' ?s' f\ x \cup \bigcup \{sources\ cs' ?vs' ?s' f\ w \mid w.$ 
    run-flow  $cs' ?vs' ?s' f: dom\ w \rightsquigarrow dom\ x \wedge w \in X\}$ 
    (is  $- \in ?A \cup ?B$ )
  moreover {
    assume  $y \in ?A$ 
    hence tags  $cs\ vs\ s\ f\ y \subseteq tags\ ?cs\ vs\ s\ f\ x$ 
      using  $G$  and  $H$  by simp
  }
  moreover {
    assume  $y \in ?B$ 
    hence tags  $cs\ vs\ s\ f\ y \subseteq \bigcup \{tags\ ?cs\ vs\ s\ f\ w \mid w.$ 
      run-flow  $?cs\ vs\ s\ f: dom\ w \rightsquigarrow dom\ x \wedge w \in X\}$ 
      using  $G$  and  $H$  by (auto simp: run-flow-append)
  }
  ultimately show ?thesis
    using  $H$  by (simp only: append-assoc [symmetric] tags.simps, auto)
  qed
qed

```

lemma *tags-member-2*:

assumes

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

assumes

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

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

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

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

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

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

proof (*subst (asm) sources-aux.simps, split com-flow.split-asm*)

fix $z a$

assume $c = z ::= a$ **and** $y \in \text{sources-aux } cs' \text{ ?vs' ?s' } f x$

moreover from this have $\text{tags } cs \text{ vs } s \text{ } f y \subseteq \text{tags-aux } ?cs \text{ vs } s \text{ } f x$

using A **by** *simp*

ultimately show *?thesis*

by (*simp only: append-assoc [symmetric] tags-aux.simps, auto*)

next

fix z

assume $c = \text{IN } z$ **and** $y \in \text{sources-aux } cs' \text{ ?vs' ?s' } f x$

moreover from this have $\text{tags } cs \text{ vs } s \text{ } f y \subseteq \text{tags-aux } ?cs \text{ vs } s \text{ } f x$

using B **by** *simp*

ultimately show *?thesis*

by (*simp only: append-assoc [symmetric] tags-aux.simps, auto*)

next

fix z

assume $c = \text{OUT } z$ **and** $y \in \text{sources-aux } cs' \text{ ?vs' ?s' } f x$

moreover from this have $\text{tags } cs \text{ vs } s \text{ } f y \subseteq \text{tags-aux } ?cs \text{ vs } s \text{ } f x$

using C **by** *simp*

ultimately show *?thesis*

by (*simp only: append-assoc [symmetric] tags-aux.simps, auto*)

next

fix X

assume $F: c = \langle X \rangle$

assume $y \in \text{sources-aux } cs' \text{ ?vs' ?s' } f x \cup \bigcup \{\text{sources } cs' \text{ ?vs' ?s' } f w \mid w.$

$\text{run-flow } cs' \text{ ?vs' ?s' } f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X\}$

(is - \in ? $A \cup ?B$)
moreover {
 assume $y \in ?A$
 hence $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-aux } ?cs \text{ vs } s f x$
 using D **and** F **by** *simp*
 }
moreover {
 assume $y \in ?B$
 hence $\text{tags } cs \text{ vs } s f y \subseteq \bigcup \{ \text{tags } ?cs \text{ vs } s f w \mid w. \text{run-flow } ?cs \text{ vs } s f: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$
 using E **and** F **by** (*auto simp: run-flow-append*)
 }
ultimately show ?*thesis*
using F **by** (*simp only: append-assoc [symmetric] tags-aux.simps, auto*)
qed

lemma *tags-member*:

$y \in \text{sources } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags } (cs @ cs') \text{ vs } s f x$
and *tags-aux-member*:
 $y \in \text{sources-aux } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-aux } (cs @ cs') \text{ vs } s f x$
by (*induction cs' vs s f x and cs' vs s f x rule: tags-induct,*
erule-tac [3] tags-member-2, erule tags-member-1, simp-all)

lemma *tags-out-member-1*:

assumes
 $A: \bigwedge z a. c = (z ::= a :: \text{com-flow}) \implies$
 $y \in \text{sources-out } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } (cs @ cs') \text{ vs } s f x$
 (is $\bigwedge - . - \implies - \in \text{sources-out } - ?vs' ?s' - - \implies$
 $- \subseteq \text{tags-out } ?cs - - -$)
assumes
 $B: \bigwedge z. c = (IN z :: \text{com-flow}) \implies$
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } ?cs \text{ vs } s f x$ **and**
 $C: \bigwedge z. c = (OUT z :: \text{com-flow}) \implies$
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } ?cs \text{ vs } s f x$ **and**
 $D: \bigwedge X. c = \langle X \rangle \implies$
 $y \in \text{sources-out } cs' ?vs' ?s' f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } ?cs \text{ vs } s f x$
shows $y \in \text{sources-out } (cs' @ [c]) ?vs' ?s' f x \implies$
 $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } (cs @ cs' @ [c]) \text{ vs } s f x$
proof (*subst (asm) sources-out.simps, split com-flow.split-asm*)
fix $z a$
assume $c = z ::= a$ **and** $y \in \text{sources-out } cs' ?vs' ?s' f x$
moreover from this have $\text{tags } cs \text{ vs } s f y \subseteq \text{tags-out } ?cs \text{ vs } s f x$

```

    using A by simp
  ultimately show ?thesis
    by (simp only: append-assoc [symmetric] tags-out.simps, auto)
next
fix z
assume c = IN z and y ∈ sources-out cs' ?vs' ?s' f x
moreover from this have tags cs vs s f y ⊆ tags-out ?cs vs s f x
  using B by simp
ultimately show ?thesis
  by (simp only: append-assoc [symmetric] tags-out.simps, auto)
next
fix z
assume E: c = OUT z
show y ∈ sources-out cs' ?vs' ?s' f x ∪
  (if z = x then sources cs' ?vs' ?s' f x else {}) ⇒ ?thesis
  (is - ∈ ?A ∪ (if - then ?B else -) ⇒ -)
proof (split if-split-asm)
  assume z = x and y ∈ ?A ∪ ?B
  moreover {
    assume y ∈ ?A
    hence tags cs vs s f y ⊆ tags-out ?cs vs s f x
      using C and E by simp
  }
  moreover {
    assume y ∈ ?B
    hence tags cs vs s f y ⊆ tags ?cs vs s f x
      by (rule tags-member)
  }
  ultimately show ?thesis
    using E by (simp only: append-assoc [symmetric] tags-out.simps, auto)
next
assume y ∈ ?A ∪ {}
moreover from this have tags cs vs s f y ⊆ tags-out ?cs vs s f x
  using C and E by simp
ultimately show ?thesis
  using E by (simp only: append-assoc [symmetric] tags-out.simps, auto)
qed
next
fix X
assume E: c = ⟨X⟩
assume y ∈ sources-out cs' ?vs' ?s' f x ∪ ∪ {sources cs' ?vs' ?s' f w | w.
  run-flow cs' ?vs' ?s' f: dom w ↪ dom x ∧ w ∈ X}
  (is - ∈ ?A ∪ ?B)
moreover {
  assume y ∈ ?A
  hence tags cs vs s f y ⊆ tags-out ?cs vs s f x
    using D and E by simp
}
moreover {

```

assume $y \in ?B$
hence $\text{tags } cs \text{ vs } s \text{ f } y \subseteq \bigcup \{ \text{tags } ?cs \text{ vs } s \text{ f } w \mid w. \\ \text{run-flow } ?cs \text{ vs } s \text{ f}: \text{dom } w \rightsquigarrow \text{dom } x \wedge w \in X \}$
by (*auto dest: tags-member simp: run-flow-append*)
}
ultimately show *?thesis*
using E **by** (*simp only: append-assoc [symmetric] tags-out.simps, auto*)
qed

lemma *tags-out-member:*

$y \in \text{sources-out } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s \text{ f}) f x \implies \\ \text{tags } cs \text{ vs } s \text{ f } y \subseteq \text{tags-out } (cs @ cs') \text{ vs } s \text{ f } x$
by (*induction cs' vs s f x rule: tags-out.induct, erule tags-out-member-1, simp-all*)

lemma *tags-suffix-1:*

assumes

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

assumes

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

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

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

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

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

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

shows $\text{tags } (cs' @ [c]) ?vs' ?s' f x =$

$\{p. \text{ case } p \text{ of } (w, n) \Rightarrow (w, \text{ length } [c \leftarrow \text{cs}. c = \text{IN } w] + n)$
 $\in \text{ tags } (cs @ cs' @ [c]) \text{ vs } s f x\}$
 $(\text{is } - = \{p. \text{ case } p \text{ of } (w, n) \Rightarrow ?P w n c\})$
apply (*subst tags.simps*)
apply (*split com-flow.split*)
apply (*rule conjI*)
subgoal
proof –
show $\forall z a. c = z ::= a \longrightarrow (\text{if } z = x$
 $\text{ then } \text{tags-aux } cs' ?vs' ?s' f x \cup \bigcup \{ \text{tags } cs' ?vs' ?s' f y \mid y.$
 $\text{ run-flow } cs' ?vs' ?s' f: \text{ dom } y \rightsquigarrow \text{ dom } x \wedge y \in \text{ avars } a\}$
 $\text{ else } \text{tags } cs' ?vs' ?s' f x) =$
 $\{(w, n). ?P w n c\}$
 $(\text{is } \forall - a. - \longrightarrow (\text{if } - \text{ then } ?A \cup ?F a \text{ else } ?B) = -)$
apply *clarify*
apply (*split if-split*)
apply (*rule conjI*)
subgoal for $z a$
proof
assume $H: z = x$ **and** $I: c = z ::= a$
hence $?A = \{(w, n). (w, \text{ length } [c \leftarrow \text{cs}. c = \text{IN } w] + n)$
 $\in \text{ tags-aux } (cs @ cs') \text{ vs } s f x\}$
using A **by** *simp*
moreover have $\forall y. \text{ tags } cs' ?vs' ?s' f y = \{(w, n).$
 $(w, \text{ length } [c \leftarrow \text{cs}. c = \text{IN } w] + n) \in \text{ tags } (cs @ cs') \text{ vs } s f y\}$
using B **and** H **and** I **by** *simp*
hence $?F a = \{(w, n). (w, \text{ length } [c \leftarrow \text{cs}. c = \text{IN } w] + n)$
 $\in \bigcup \{ \text{tags } (cs @ cs') \text{ vs } s f y \mid y.$
 $\text{ run-flow } cs' ?vs' ?s' f: \text{ dom } y \rightsquigarrow \text{ dom } x \wedge y \in \text{ avars } a\}\}$
by *blast*
ultimately show $?A \cup ?F a = \{(w, n). ?P w n (z ::= a)\}$
using H **by** (*subst append-assoc [symmetric], subst tags.simps,*
auto simp: run-flow-append)
qed
subgoal for $z a$
proof
assume $z \neq x$ **and** $c = z ::= a$
moreover from *this* **have** $?B = \{(w, n).$
 $(w, \text{ length } [c \leftarrow \text{cs}. c = \text{IN } w] + n) \in \text{ tags } (cs @ cs') \text{ vs } s f x\}$
using C **by** *simp*
ultimately show $?B = \{(w, n). ?P w n (z ::= a)\}$
by (*subst append-assoc [symmetric], subst tags.simps, simp*)
qed
done
qed
apply (*rule conjI*)
subgoal
proof –
show $\forall z. c = \text{IN } z \longrightarrow (\text{if } z = x$

then insert $(x, \text{length } [c \leftarrow cs'. c = IN x])$ (tags-aux $cs' ?vs' ?s' f x$)
 else tags $cs' ?vs' ?s' f x$ =
 $\{(w, n). ?P w n c\}$
 (is $\forall -. - \longrightarrow$ (if - then insert $?p ?A$ else $?B$) = -)
 apply clarify
 apply (split if-split)
 apply (rule conjI)
 subgoal for z
 proof
 assume $z = x$ and $c = IN z$
 moreover from this have $?A = \{(w, n).$
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags-aux } (cs @ cs') \text{ vs } s f x\}$
 using D by simp
 ultimately show insert $?p ?A = \{(w, n). ?P w n (IN z)\}$
 by (subst append-assoc [symmetric], subst tags.simps, auto)
 qed
 subgoal for z
 proof
 assume $z \neq x$ and $c = IN z$
 moreover from this have $?B = \{(w, n).$
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags } (cs @ cs') \text{ vs } s f x\}$
 using E by simp
 ultimately show $?B = \{(w, n). ?P w n (IN z)\}$
 by (subst append-assoc [symmetric], subst tags.simps, simp)
 qed
 done
 qed
 apply (rule conjI)
 subgoal by (subst append-assoc [symmetric], subst tags.simps, simp add: F)
 subgoal
 proof -
 show $\forall X. c = \langle X \rangle \longrightarrow$
 tags $cs' ?vs' ?s' f x \cup \bigcup \{\text{tags } cs' ?vs' ?s' f y \mid y.$
 $\text{run-flow } cs' ?vs' ?s' f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X\} =$
 $\{(w, n). ?P w n c\}$
 (is $\forall X. - \longrightarrow ?A \cup ?F X = -$)
 proof clarify
 fix X
 assume $H: c = \langle X \rangle$
 hence $?A = \{(w, n). (w, \text{length } [c \leftarrow cs. c = IN w] + n)$
 $\in \text{tags } (cs @ cs') \text{ vs } s f x\}$
 using G by simp
 moreover have $\forall y. \text{tags } cs' ?vs' ?s' f y = \{(w, n).$
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags } (cs @ cs') \text{ vs } s f y\}$
 using G and H by simp
 hence $?F X = \{(w, n). (w, \text{length } [c \leftarrow cs. c = IN w] + n)$
 $\in \bigcup \{\text{tags } (cs @ cs') \text{ vs } s f y \mid y.$
 $\text{run-flow } cs' ?vs' ?s' f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X\}\}$
 by blast

ultimately show $?A \cup ?F X = \{(w, n). ?P w n (\langle X \rangle)\}$
by (*subst append-assoc [symmetric], subst tags.simps,*
auto simp: run-flow-append)

qed
qed
done

lemma *tags-suffix-2:*

assumes

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

assumes

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

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

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

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

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

apply (*subst tags-aux.simps*)

apply (*split com-flow.split*)

apply (*rule conjI*)

defer

apply (*rule conjI*)

defer

apply (*rule conjI*)

defer

subgoal

proof –

show $\forall X. c = \langle X \rangle \longrightarrow$
tags-aux cs' ?vs' ?s' f x $\cup \cup$ {tags cs' ?vs' ?s' f y | y.
run-flow cs' ?vs' ?s' f: dom y \rightsquigarrow dom x \wedge y \in X} =
{(w, n). ?P w n c}

(is $\forall X. - \longrightarrow ?A \cup ?F X = -$)

proof clarify

fix X

assume $F: c = \langle X \rangle$

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

using D **by** *simp*

moreover have $\forall y. \text{tags } cs' ?vs' ?s' f y = \{(w, n).$

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

using E **and** F **by** *simp*

hence $?F X = \{(w, n). (w, \text{length } [c \leftarrow cs. c = IN w] + n)$

$\in \bigcup \{\text{tags } (cs @ cs') \text{ vs } s f y \mid y.$

$\text{run-flow } cs' ?vs' ?s' f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X\}$

by *blast*

ultimately show $?A \cup ?F X = \{(w, n). ?P w n (\langle X \rangle)\}$

by (*subst append-assoc [symmetric]*, *subst tags-aux.simps*,
auto simp: run-flow-append)

qed

qed

by (*subst append-assoc [symmetric]*, *subst tags-aux.simps*, *simp add: A B C*) $+$

lemma tags-suffix:

$\text{tags } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x = \{(w, n).$

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

and tags-aux-suffix:

$\text{tags-aux } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x = \{(w, n).$

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

by (*induction cs' vs s f x and cs' vs s f x rule: tags-induct*,

erule-tac [3] tags-suffix-2, erule tags-suffix-1, simp-all

add: tags-ubound tags-aux-ubound)

lemma tags-out-suffix-1:

assumes

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

$\text{tags-out } cs' (vs @ \text{in-flow } cs \text{ vs } f) (\text{run-flow } cs \text{ vs } s f) f x =$

$\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$

$\in \text{tags-out } (cs @ cs') \text{ vs } s f x\}$

(is $\bigwedge - - \implies \text{tags-out } - ?vs' ?s' - - = -$)

assumes

$B: \bigwedge z. c = (IN z :: \text{com-flow}) \implies$

$\text{tags-out } cs' ?vs' ?s' f x =$

$\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$

$\in \text{tags-out } (cs @ cs') \text{ vs } s f x\}$ **and**

$C: \bigwedge z. c = (OUT z :: \text{com-flow}) \implies$

$\text{tags-out } cs' ?vs' ?s' f x =$

$\{p. \text{case } p \text{ of } (w, n) \Rightarrow (w, \text{length } [c \leftarrow cs. c = IN w] + n)$

$\in \text{tags-out } (cs @ cs') \text{ vs } s f x\}$ **and**

$D: \bigwedge X. c = \langle X \rangle \implies$

$tags\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x =$
 $\{p.\ case\ p\ of\ (w, n) \Rightarrow (w, length\ [c\leftarrow\ cs.\ c = IN\ w] + n)$
 $\in\ tags\text{-}out\ (cs\ @\ cs')\ vs\ s\ f\ x\}$
shows $tags\text{-}out\ (cs'\ @\ [c])\ ?vs'\ ?s'\ f\ x =$
 $\{p.\ case\ p\ of\ (w, n) \Rightarrow (w, length\ [c\leftarrow\ cs.\ c = IN\ w] + n)$
 $\in\ tags\text{-}out\ (cs\ @\ cs'\ @\ [c])\ vs\ s\ f\ x\}$
(is $- = \{p.\ case\ p\ of\ (w, n) \Rightarrow ?P\ w\ n\ c\}$)
apply $(subst\ tags\text{-}out.\ simps)$
apply $(split\ com\text{-}flow.\ split)$
apply $(rule\ conjI)$
defer
apply $(rule\ conjI)$
defer
subgoal
proof
show $\forall z.\ c = OUT\ z \longrightarrow$
 $tags\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x \cup$
 $(if\ z = x\ then\ tags\ cs'\ ?vs'\ ?s'\ f\ x\ else\ \{\}) =$
 $\{(w, n).\ ?P\ w\ n\ c\}$
(is $\forall -. - \longrightarrow ?A \cup (if\ -\ then\ ?B\ else\ -) = -)$
apply $clarify$
apply $(split\ if\text{-}split)$
apply $(rule\ conjI)$
subgoal for z
proof
assume $c = OUT\ z$ **and** $z = x$
moreover from this have $?A = \{p.\ case\ p\ of\ (w, n) \Rightarrow$
 $(w, length\ [c\leftarrow\ cs.\ c = IN\ w] + n) \in\ tags\text{-}out\ (cs\ @\ cs')\ vs\ s\ f\ x\}$
using C **by** $simp$
moreover have $?B = \{(w, n).\$
 $(w, length\ [c\leftarrow\ cs.\ c = IN\ w] + n) \in\ tags\ (cs\ @\ cs')\ vs\ s\ f\ x\}$
by $(rule\ tags\text{-}suffix)$
ultimately show $?A \cup ?B = \{(w, n).\ ?P\ w\ n\ (OUT\ z)\}$
by $(subst\ append\text{-}assoc\ [symmetric],\ subst\ tags\text{-}out.\ simps,\ auto)$
qed
subgoal for z
proof
assume $c = OUT\ z$ **and** $z \neq x$
moreover from this have $?A = \{p.\ case\ p\ of\ (w, n) \Rightarrow$
 $(w, length\ [c\leftarrow\ cs.\ c = IN\ w] + n) \in\ tags\text{-}out\ (cs\ @\ cs')\ vs\ s\ f\ x\}$
using C **by** $simp$
ultimately show $?A \cup \{\} = \{(w, n).\ ?P\ w\ n\ (OUT\ z)\}$
by $(subst\ append\text{-}assoc\ [symmetric],\ subst\ tags\text{-}out.\ simps,\ simp)$
qed
done
next
show $\forall X.\ c = \langle X \rangle \longrightarrow$
 $tags\text{-}out\ cs'\ ?vs'\ ?s'\ f\ x \cup \bigcup\ \{tags\ cs'\ ?vs'\ ?s'\ f\ y\ |\ y.\$
 $run\text{-}flow\ cs'\ ?vs'\ ?s'\ f:\ dom\ y \rightsquigarrow\ dom\ x \wedge y \in X\} =$

$\{(w, n). ?P w n c\}$
(is $\forall X. - \longrightarrow ?A \cup ?F X = -)$
proof clarify
fix X
assume $c = \langle X \rangle$
hence $?A = \{(w, n). (w, \text{length } [c \leftarrow cs. c = IN w] + n)$
 $\in \text{tags-out } (cs @ cs') vs s f x\}$
using D **by** *simp*
moreover have $\forall y. \text{tags } cs' ?vs' ?s' f y = \{(w, n).$
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags } (cs @ cs') vs s f y\}$
by (*blast intro!: tags-suffix*)
hence $?F X = \{(w, n). (w, \text{length } [c \leftarrow cs. c = IN w] + n)$
 $\in \bigcup \{\text{tags } (cs @ cs') vs s f y \mid y.$
 $\text{run-flow } cs' ?vs' ?s' f: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in X\}\}$
by *blast*
ultimately show $?A \cup ?F X = \{(w, n). ?P w n (\langle X \rangle)\}$
by (*subst append-assoc [symmetric], subst tags-out.simps,*
auto simp: run-flow-append)
qed
qed
by (*subst append-assoc [symmetric], subst tags-out.simps, simp add: A B*) $+$

lemma tags-out-suffix:
 $\text{tags-out } cs' (vs @ \text{in-flow } cs vs f) (\text{run-flow } cs vs s f) f x = \{(w, n).$
 $(w, \text{length } [c \leftarrow cs. c = IN w] + n) \in \text{tags-out } (cs @ cs') vs s f x\}$
by (*induction cs' vs s f x rule: tags-out.induct,*
erule tags-out-suffix-1, simp-all add: tags-out-ubound)

lemma sources-aux-rhs:
assumes
 $A: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } (\text{flow } cfs @ cs') vs_1 s_1 f x)\}$
 $(\text{is } - \subseteq \{-. - = - (\subseteq \text{sources-aux } (?cs @ -) - - -)\})$
assumes
 $B: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{\text{tags-aux } (?cs @ cs') vs_1 s_1 f x \mid x. x \in S\})$ **and**
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$ **and**
 $D: \text{ok-flow-aux-2 } s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs$
shows $S \subseteq \{x. s_2 = t_2 (\subseteq \text{sources-aux } cs' vs_2 s_2 f x)\}$
proof clarify
fix $x y$
assume $E: y \in \text{sources-aux } cs' vs_2 s_2 f x$
moreover have $F: s_2 = \text{run-flow } ?cs vs_1 s_1 f$
using C **by** (*rule small-steps-run-flow*)
moreover have $G: vs_2 = vs_1 @ \text{in-flow } ?cs vs_1 f$
using C **by** (*rule small-steps-in-flow*)
ultimately have $\text{sources } ?cs vs_1 s_1 f y \subseteq \text{sources-aux } (?cs @ cs') vs_1 s_1 f x$
by (*blast dest: sources-aux-member*)
moreover assume $H: x \in S$

ultimately have $s_1 = t_1$ (\subseteq *sources* $?cs$ vs_1 s_1 f y)
using A *by blast*
moreover have $tags$ $?cs$ vs_1 s_1 f y \subseteq $tags$ -aux ($?cs$ @ cs') vs_1 s_1 f x
using E **and** F **and** G *by* (*blast dest: tags-aux-member*)
hence $tags$ $?cs$ vs_1 s_1 f y \subseteq \bigcup { $tags$ -aux ($?cs$ @ cs') vs_1 s_1 f x | $x. x \in S$ }
using H *by blast*
with B **have** $f = f'$ (\subseteq $vs_1, vs_1', tags$ $?cs$ vs_1 s_1 f y)
by (*rule eq-streams-subset*)
ultimately show s_2 $y = t_2$ y
using D [*rule-format, of {y}*] *by simp*
qed

lemma *sources-rhs:*

assumes

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

(**is** \subseteq { \cdot . $\cdot = \cdot$ (\subseteq *sources* ($?cs$ @ \cdot) $\cdot \cdot \cdot$)}

assumes

$B: f = f'$ (\subseteq vs_1, vs_1' ,

\bigcup { $tags$ ($?cs$ @ cs') vs_1 s_1 f x | $x. x \in S$ }) **and**

$C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs\} (c_2, s_2, f, vs_2, ws_2)$ **and**

$D: ok$ -flow-aux-2 s_1 s_2 t_1 t_2 f f' vs_1 vs_1' $?cs$

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

proof *clarify*

fix x y

assume $E: y \in$ *sources* cs' vs_2 s_2 f x

moreover have $F: s_2 =$ *run-flow* $?cs$ vs_1 s_1 f

using C *by* (*rule small-stepsl-run-flow*)

moreover have $G: vs_2 = vs_1$ @ *in-flow* $?cs$ vs_1 f

using C *by* (*rule small-stepsl-in-flow*)

ultimately have *sources* $?cs$ vs_1 s_1 f y \subseteq *sources* ($?cs$ @ cs') vs_1 s_1 f x

by (*blast dest: sources-member*)

moreover assume $H: x \in S$

ultimately have $s_1 = t_1$ (\subseteq *sources* $?cs$ vs_1 s_1 f y)

using A *by blast*

moreover have $tags$ $?cs$ vs_1 s_1 f y \subseteq $tags$ ($?cs$ @ cs') vs_1 s_1 f x

using E **and** F **and** G *by* (*blast dest: tags-member*)

hence $tags$ $?cs$ vs_1 s_1 f y \subseteq \bigcup { $tags$ ($?cs$ @ cs') vs_1 s_1 f x | $x. x \in S$ }

using H *by blast*

with B **have** $f = f'$ (\subseteq $vs_1, vs_1', tags$ $?cs$ vs_1 s_1 f y)

by (*rule eq-streams-subset*)

ultimately show s_2 $y = t_2$ y

using D [*rule-format, of {y}*] *by simp*

qed

lemma *sources-out-rhs:*

assumes

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

(**is** \subseteq { \cdot . $\cdot = \cdot$ (\subseteq *sources-out* ($?cs$ @ \cdot) $\cdot \cdot \cdot$)}

assumes

$B: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags-out (?cs @ cs') vs_1 s_1 f x \mid x. x \in S\})$ **and**
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs\} (c_2, s_2, f, vs_2, ws_2)$ **and**
 $D: ok-flow-aux-2 s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs$
shows $S \subseteq \{x. s_2 = t_2 (\subseteq sources-out cs' vs_2 s_2 f x)\}$
proof clarify
fix $x y$
assume $E: y \in sources-out cs' vs_2 s_2 f x$
moreover have $F: s_2 = run-flow ?cs vs_1 s_1 f$
using C **by** (rule small-stepsl-run-flow)
moreover have $G: vs_2 = vs_1 @ in-flow ?cs vs_1 f$
using C **by** (rule small-stepsl-in-flow)
ultimately have $sources ?cs vs_1 s_1 f y \subseteq sources-out (?cs @ cs') vs_1 s_1 f x$
by (blast dest: sources-out-member)
moreover assume $H: x \in S$
ultimately have $s_1 = t_1 (\subseteq sources ?cs vs_1 s_1 f y)$
using A **by** blast
moreover have $tags ?cs vs_1 s_1 f y \subseteq tags-out (?cs @ cs') vs_1 s_1 f x$
using E **and** F **and** G **by** (blast dest: tags-out-member)
hence $tags ?cs vs_1 s_1 f y \subseteq \bigcup \{tags-out (?cs @ cs') vs_1 s_1 f x \mid x. x \in S\}$
using H **by** blast
with B **have** $f = f' (\subseteq vs_1, vs_1', tags ?cs vs_1 s_1 f y)$
by (rule eq-streams-subset)
ultimately show $s_2 y = t_2 y$
using D [rule-format, of $\{y\}$] **by** simp
qed

lemma tags-aux-rhs:

assumes
 $A: S \subseteq \{x. s_1 = t_1 (\subseteq sources-aux (flow cfs @ cs') vs_1 s_1 f x)\}$
 $(is - \subseteq \{- . - - (\subseteq sources-aux (?cs @ -) - - - \})$
assumes
 $B: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags-aux (?cs @ cs') vs_1 s_1 f x \mid x. x \in S\})$ **and**
 $C: (c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs\} (c_2, s_2, f, vs_2, ws_2)$ **and**
 $D: (c_1', t_1, f', vs_1', ws_1') \rightarrow^*\{cfs\} (c_2', t_2, f', vs_2', ws_2')$ **and**
 $E: ok-flow-aux-1 c_1 c_2 c_2' s_1 t_1 t_2 f f' vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs$
shows $f = f' (\subseteq vs_2, vs_2', \bigcup \{tags-aux cs' vs_2 s_2 f x \mid x. x \in S\})$
proof (subst eq-streams-def, clarify)
fix $x y n$
have $F: vs_2 = vs_1 @ drop (length vs_1) vs_2$
using small-stepsl-steps [OF C] **by** (rule small-steps-in-flow)
have $G: vs_2' = vs_1' @ drop (length vs_1') vs_2'$
using D **by** (rule small-steps-in-flow)
assume $(y, n) \in tags-aux cs' vs_2 s_2 f x$
moreover have $s_2 = run-flow ?cs vs_1 s_1 f$
using C **by** (rule small-stepsl-run-flow)
moreover have $H: vs_2 = vs_1 @ in-flow ?cs vs_1 f$

using C **by** (*rule small-steps-l-in-flow*)
ultimately have $I: (y, \text{length } [c \leftarrow ?cs. c = IN\ y] + n)$
 $\in \text{tags-aux } (?cs @ cs')\ vs_1\ s_1\ f\ x$
(is $(-, ?k + -) \in -$
by (*simp add: tags-aux-suffix*)
let $?m = \text{Suc } (\text{Max } \{k. k \leq \text{length } (?cs @ cs') \wedge$
 $\text{length } [c \leftarrow \text{take } k\ (?cs @ cs').\ c = IN\ y] \leq ?k + n\})$
have $J: y \in \text{sources-aux } (\text{drop } ?m\ (?cs @ cs'))$
 $(vs_1 @ \text{in-flow } (\text{take } ?m\ (?cs @ cs'))\ vs_1\ f)$
 $(\text{run-flow } (\text{take } ?m\ (?cs @ cs'))\ vs_1\ s_1\ f)\ f\ x$
using I **by** (*auto dest: tags-aux-sources-aux*)
hence $\text{sources } (\text{take } ?m\ (?cs @ cs'))\ vs_1\ s_1\ f\ y \subseteq$
 $\text{sources-aux } (\text{take } ?m\ (?cs @ cs') @ \text{drop } ?m\ (?cs @ cs'))\ vs_1\ s_1\ f\ x$
by (*rule sources-aux-member*)
moreover have $K: \text{length } ?cs \leq ?m$
by (*rule le-SucI, rule Max-ge, simp-all*)
ultimately have
 $\text{sources } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y \subseteq$
 $\text{sources-aux } (?cs @ cs')\ vs_1\ s_1\ f\ x$
by *simp*
moreover have
 $\text{sources-aux } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y \subseteq$
 $\text{sources } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y$
by (*rule sources-aux-sources*)
moreover have $\text{sources-aux } ?cs\ vs_1\ s_1\ f\ y \subseteq$
 $\text{sources-aux } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y$
by (*rule sources-aux-append*)
moreover assume $L: x \in S$
hence $s_1 = t_1 (\subseteq \text{sources-aux } (?cs @ cs')\ vs_1\ s_1\ f\ x)$
using A **by** *blast*
ultimately have $M: s_1 = t_1 (\subseteq \text{sources-aux } ?cs\ vs_1\ s_1\ f\ y)$
by *blast*
have $\text{tags } (\text{take } ?m\ (?cs @ cs'))\ vs_1\ s_1\ f\ y \subseteq$
 $\text{tags-aux } (\text{take } ?m\ (?cs @ cs') @ \text{drop } ?m\ (?cs @ cs'))\ vs_1\ s_1\ f\ x$
using J **by** (*rule tags-aux-member*)
hence $\text{tags } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y \subseteq$
 $\text{tags-aux } (?cs @ cs')\ vs_1\ s_1\ f\ x$
using K **by** *simp*
moreover have
 $\text{tags-aux } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y \subseteq$
 $\text{tags } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y$
by (*rule tags-aux-tags*)
moreover have $\text{tags-aux } ?cs\ vs_1\ s_1\ f\ y \subseteq$
 $\text{tags-aux } (?cs @ \text{take } (?m - \text{length } ?cs)\ cs')\ vs_1\ s_1\ f\ y$
by (*rule tags-aux-append*)
ultimately have $\text{tags-aux } ?cs\ vs_1\ s_1\ f\ y \subseteq$
 $\bigcup \{\text{tags-aux } (?cs @ cs')\ vs_1\ s_1\ f\ x \mid x. x \in S\}$
using L **by** *blast*
with B **have** $f = f' (\subseteq vs_1, vs_1', \text{tags-aux } ?cs\ vs_1\ s_1\ f\ y)$

by (rule eq-streams-subset)
hence $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p = y] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p = y]$
using E [rule-format, of $\{y\}$] **and** M **by** *simp*
hence $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p = y] =$
 $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p = y]$
by (drule-tac arg-cong [where $f = \text{length}$],
 $\text{subst } (\text{asm}) (1\ 2) \text{ length-map}$)
hence $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p = y] = ?k \wedge$
 $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p = y] = ?k$
using H **by** (*simp add: in-flow-length*)
moreover $\text{have } f\ y\ (\text{length } [p \leftarrow vs_1. \text{fst } p = y] + ?k + n) =$
 $f'\ y\ (\text{length } [p \leftarrow vs_1'. \text{fst } p = y] + ?k + n)$
using B **and** I **and** L **by** (*fastforce simp: eq-streams-def ac-simps*)
ultimately $\text{show } f\ y\ (\text{length } [p \leftarrow vs_2. \text{fst } p = y] + n) =$
 $f'\ y\ (\text{length } [p \leftarrow vs_2'. \text{fst } p = y] + n)$
by (*subst F, subst G, simp*)
qed

lemma *tags-rhs*:

assumes

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

assumes

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

proof (*subst eq-streams-def, clarify*)

fix $x\ y\ n$

have $F: vs_2 = vs_1 @ \text{drop } (\text{length } vs_1) vs_2$
using *small-stepsl-steps* [OF C] **by** (*rule small-steps-in-flow*)

have $G: vs_2' = vs_1' @ \text{drop } (\text{length } vs_1') vs_2'$

using D **by** (*rule small-steps-in-flow*)

assume $(y, n) \in \text{tags } cs'\ vs_2\ s_2\ f\ x$

moreover $\text{have } s_2 = \text{run-flow } ?cs\ vs_1\ s_1\ f$

using C **by** (*rule small-stepsl-run-flow*)

moreover $\text{have } H: vs_2 = vs_1 @ \text{in-flow } ?cs\ vs_1\ f$

using C **by** (*rule small-stepsl-in-flow*)

ultimately $\text{have } I: (y, \text{length } [c \leftarrow ?cs. c = IN\ y] + n)$

$\in \text{tags } (?cs @ cs') vs_1\ s_1\ f\ x$

$(\text{is } (-, ?k + -) \in -)$

by (*simp add: tags-suffix*)

let $?m = \text{Suc } (\text{Max } \{k. k \leq \text{length } (?cs @ cs') \wedge$
 $\text{length } [c \leftarrow \text{take } k\ (?cs @ cs'). c = IN\ y] \leq ?k + n\})$

have $J: y \in \text{sources } (\text{drop } ?m\ (?cs @ cs'))$

$(vs_1 @ \text{in-flow } (\text{take } ?m\ (?cs @ cs')) vs_1\ f)$

$(\text{run-flow } (\text{take } ?m \text{ } (?cs \text{ @ } cs')) \text{ } vs_1 \text{ } s_1 \text{ } f) \text{ } f \text{ } x$
using I **by** $(\text{auto dest: tags-sources})$
hence $\text{sources } (\text{take } ?m \text{ } (?cs \text{ @ } cs')) \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{sources } (\text{take } ?m \text{ } (?cs \text{ @ } cs') \text{ @ drop } ?m \text{ } (?cs \text{ @ } cs')) \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x$
by $(\text{rule sources-member})$
moreover have $K: \text{length } ?cs \leq ?m$
by $(\text{rule le-SucI, rule Max-ge, simp-all})$
ultimately have
 $\text{sources } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{sources } (?cs \text{ @ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x$
by simp
moreover have
 $\text{sources-aux } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{sources } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y$
by $(\text{rule sources-aux-sources})$
moreover have $\text{sources-aux } ?cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{sources-aux } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y$
by $(\text{rule sources-aux-append})$
moreover assume $L: x \in S$
hence $s_1 = t_1 (\subseteq \text{sources } (?cs \text{ @ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x)$
using A **by** blast
ultimately have $M: s_1 = t_1 (\subseteq \text{sources-aux } ?cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y)$
by blast
have $\text{tags } (\text{take } ?m \text{ } (?cs \text{ @ } cs')) \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{tags } (\text{take } ?m \text{ } (?cs \text{ @ } cs') \text{ @ drop } ?m \text{ } (?cs \text{ @ } cs')) \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x$
using J **by** $(\text{rule tags-member})$
hence $\text{tags } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{tags } (?cs \text{ @ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x$
using K **by** simp
moreover have
 $\text{tags-aux } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{tags } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y$
by $(\text{rule tags-aux-tags})$
moreover have $\text{tags-aux } ?cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\text{tags-aux } (?cs \text{ @ take } (?m - \text{length } ?cs) \text{ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y$
by $(\text{rule tags-aux-append})$
ultimately have $\text{tags-aux } ?cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y \subseteq$
 $\bigcup \{ \text{tags } (?cs \text{ @ } cs') \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } x \mid x. x \in S \}$
using L **by** blast
with B **have** $f = f' (\subseteq vs_1, vs_1', \text{tags-aux } ?cs \text{ } vs_1 \text{ } s_1 \text{ } f \text{ } y)$
by $(\text{rule eq-streams-subset})$
hence $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) \text{ } vs_2. \text{fst } p = y] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') \text{ } vs_2'. \text{fst } p = y]$
using E $[\text{rule-format, of } \{y\}]$ **and** M **by** simp
hence $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1) \text{ } vs_2. \text{fst } p = y] =$
 $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1') \text{ } vs_2'. \text{fst } p = y]$
by $(\text{drule-tac arg-cong } [\text{where } f = \text{length}],$
 $\text{subst } (\text{asm}) \text{ } (1 \ 2) \text{ length-map})$
hence $\text{length } [p \leftarrow \text{drop } (\text{length } vs_1) \text{ } vs_2. \text{fst } p = y] = ?k \wedge$

$\text{length } [p \leftarrow \text{drop } (\text{length } vs_1') \text{ } vs_2'. \text{fst } p = y] = ?k$
using H **by** (*simp add: in-flow-length*)
moreover have $f \ y \ (\text{length } [p \leftarrow vs_1. \text{fst } p = y] + ?k + n) =$
 $f' \ y \ (\text{length } [p \leftarrow vs_1'. \text{fst } p = y] + ?k + n)$
using B **and** I **and** L **by** (*fastforce simp: eq-streams-def ac-simps*)
ultimately show $f \ y \ (\text{length } [p \leftarrow vs_2. \text{fst } p = y] + n) =$
 $f' \ y \ (\text{length } [p \leftarrow vs_2'. \text{fst } p = y] + n)$
by (*subst F, subst G, simp*)
qed

lemma *tags-out-rhs*:

assumes

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

assumes

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

proof (*subst eq-streams-def, clarify*)

fix $x \ y \ n$

have $F: vs_2 = vs_1 \ @ \ \text{drop } (\text{length } vs_1) \ vs_2$
using *small-stepsl-steps* [*OF C*] **by** (*rule small-steps-in-flow*)

have $G: vs_2' = vs_1' \ @ \ \text{drop } (\text{length } vs_1') \ vs_2'$
using D **by** (*rule small-steps-in-flow*)

assume $(y, n) \in \text{tags-out } cs' \ vs_2 \ s_2 \ f \ x$

moreover have $s_2 = \text{run-flow } ?cs \ vs_1 \ s_1 \ f$
using C **by** (*rule small-stepsl-run-flow*)

moreover have $H: vs_2 = vs_1 \ @ \ \text{in-flow } ?cs \ vs_1 \ f$
using C **by** (*rule small-stepsl-in-flow*)

ultimately have $I: (y, \text{length } [c \leftarrow ?cs. c = IN \ y] + n)$
 $\in \text{tags-out } (?cs \ @ \ cs') \ vs_1 \ s_1 \ f \ x$
 $(\text{is } (-, ?k + -) \in -)$

by (*simp add: tags-out-suffix*)

let $?m = \text{Suc } (\text{Max } \{k. k \leq \text{length } (?cs \ @ \ cs') \wedge$
 $\text{length } [c \leftarrow \text{take } k \ (?cs \ @ \ cs'). c = IN \ y] \leq ?k + n\})$

have $J: y \in \text{sources-out } (\text{drop } ?m \ (?cs \ @ \ cs'))$
 $(vs_1 \ @ \ \text{in-flow } (\text{take } ?m \ (?cs \ @ \ cs')) \ vs_1 \ f)$

$(\text{run-flow } (\text{take } ?m \ (?cs \ @ \ cs')) \ vs_1 \ s_1 \ f) \ f \ x$

using I **by** (*auto dest: tags-out-sources-out*)

hence $\text{sources } (\text{take } ?m \ (?cs \ @ \ cs')) \ vs_1 \ s_1 \ f \ y \subseteq$

$\text{sources-out } (\text{take } ?m \ (?cs \ @ \ cs') \ @ \ \text{drop } ?m \ (?cs \ @ \ cs')) \ vs_1 \ s_1 \ f \ x$

by (*rule sources-out-member*)

moreover have $K: \text{length } ?cs \leq ?m$

by (*rule le-SucI, rule Max-ge, simp-all*)

ultimately have

$\text{sources } (?cs \ @ \ \text{take } (?m - \text{length } ?cs) \ cs') \ vs_1 \ s_1 \ f \ y \subseteq$

sources-out (?cs @ cs') vs₁ s₁ f x
by *simp*
moreover have
sources-aux (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y ⊆
sources (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y
by (rule *sources-aux-sources*)
moreover have *sources-aux* ?cs vs₁ s₁ f y ⊆
sources-aux (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y
by (rule *sources-aux-append*)
moreover assume L: x ∈ S
hence s₁ = t₁ (⊆ *sources-out* (?cs @ cs') vs₁ s₁ f x)
using A **by** *blast*
ultimately have M: s₁ = t₁ (⊆ *sources-aux* ?cs vs₁ s₁ f y)
by *blast*
have *tags* (take ?m (?cs @ cs')) vs₁ s₁ f y ⊆
tags-out (take ?m (?cs @ cs') @ drop ?m (?cs @ cs')) vs₁ s₁ f x
using J **by** (rule *tags-out-member*)
hence *tags* (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y ⊆
tags-out (?cs @ cs') vs₁ s₁ f x
using K **by** *simp*
moreover have
tags-aux (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y ⊆
tags (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y
by (rule *tags-aux-tags*)
moreover have *tags-aux* ?cs vs₁ s₁ f y ⊆
tags-aux (?cs @ take (?m - length ?cs) cs') vs₁ s₁ f y
by (rule *tags-aux-append*)
ultimately have *tags-aux* ?cs vs₁ s₁ f y ⊆
⋃ {*tags-out* (?cs @ cs') vs₁ s₁ f x | x. x ∈ S}
using L **by** *blast*
with B **have** f = f' (⊆ vs₁, vs₁', *tags-aux* ?cs vs₁ s₁ f y)
by (rule *eq-streams-subset*)
hence *map fst* [p←drop (length vs₁) vs₂. *fst* p = y] =
map fst [p←drop (length vs₁') vs₂'. *fst* p = y]
using E [rule-format, of {y}] **and** M **by** *simp*
hence *length* [p←drop (length vs₁) vs₂. *fst* p = y] =
length [p←drop (length vs₁') vs₂'. *fst* p = y]
by (*drule-tac arg-cong* [where f = *length*],
subst (asm) (1 2) length-map)
hence *length* [p←drop (length vs₁) vs₂. *fst* p = y] = ?k ∧
length [p←drop (length vs₁') vs₂'. *fst* p = y] = ?k
using H **by** (*simp add: in-flow-length*)
moreover have f y (length [p←vs₁. *fst* p = y] + ?k + n) =
f' y (length [p←vs₁'. *fst* p = y] + ?k + n)
using B **and** I **and** L **by** (*fastforce simp: eq-streams-def ac-simps*)
ultimately show f y (length [p←vs₂. *fst* p = y] + n) =
f' y (length [p←vs₂'. *fst* p = y] + n)
by (*subst F, subst G, simp*)
qed

lemma *ctyping2-term-seq*:

assumes

$A: \bigwedge B Y p. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_1, p) \Rightarrow p'$ **and**
 $B: \bigwedge q B Y B' Y' p. (U, v) \models c_1 (\subseteq A, X) = \text{Some } q \implies (B, Y) = q \implies$
 $(U, v) \models c_2 (\subseteq B, Y) = \text{Some } (B', Y') \implies$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_2, p) \Rightarrow p'$ **and**
 $C: (U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (B', Y')$ **and**
 $D: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV}$
shows $\exists p'. (c_1;; c_2, p) \Rightarrow p'$

proof –

obtain B **and** Y **where**

$E: (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y)$ **and**
 $F: (U, v) \models c_2 (\subseteq B, Y) = \text{Some } (B', Y')$
using C **by** (*auto split: option.split-asm*)

obtain p' **where** $(c_1, p) \Rightarrow p'$

using A [*OF E D*] **by** *blast*

moreover obtain p'' **where** $(c_2, p') \Rightarrow p''$

using B [*OF E - F D*] **by** *blast*

ultimately show *?thesis*

by *blast*

qed

lemma *ctyping2-term-or*:

assumes

$A: \bigwedge B Y p. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_1, p) \Rightarrow p'$ **and**
 $B: (U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B', Y')$ **and**
 $C: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV}$
shows $\exists p'. (c_1 \text{ OR } c_2, p) \Rightarrow p'$

proof –

obtain B **and** Y **where** $(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y)$

using B **by** (*auto split: option.split-asm*)

thus *?thesis*

using A **and** C **by** *blast*

qed

lemma *ctyping2-term-if*:

assumes

$A: \bigwedge U' q B_1 B_2 B Y p.$
 $(U', q) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$
 $(B_1, B_2) = q \implies (U', v) \models c_1 (\subseteq B_1, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_1, p) \Rightarrow p'$ **and**
 $B: \bigwedge U' q B_1 B_2 B Y p.$
 $(U', q) = (\text{insert } (\text{Univ? } A X, \text{bvars } b) U, \models b (\subseteq A, X)) \implies$
 $(B_1, B_2) = q \implies (U', v) \models c_2 (\subseteq B_2, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_2, p) \Rightarrow p'$ **and**

$C: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y) \text{ and}$
 $D: \exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV}$
shows $\exists p'. (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, p) \Rightarrow p'$
proof –
let $?U' = \text{insert } (\text{Univ? } A \ X, \text{bvars } b) \ U$
obtain B_1 **and** B_1' **and** Y_1 **and** B_2 **and** B_2' **and** Y_2 **where**
 $E: \models b (\subseteq A, X) = (B_1, B_2) \text{ and}$
 $F: (?U', v) \models c_1 (\subseteq B_1, X) = \text{Some } (B_1', Y_1) \text{ and}$
 $G: (?U', v) \models c_2 (\subseteq B_2, X) = \text{Some } (B_2', Y_2)$
using C **by** (*auto split: option.split-asm prod.split-asm*)
obtain s **and** q **where** $p = (s, q)$
by (*cases p*)
moreover {
assume $\text{bval } b \ s$
moreover obtain p' **where** $(c_1, s, q) \Rightarrow p'$
using A [*OF - - F, of - B2 (s, q)*] **and** D **and** E **by** *auto*
ultimately have $\exists p'. (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s, q) \Rightarrow p'$
by *blast*
}
moreover {
assume $\neg \text{bval } b \ s$
moreover obtain p' **where** $(c_2, s, q) \Rightarrow p'$
using B [*OF - - G, of - B1 (s, q)*] **and** D **and** E **by** *auto*
ultimately have $\exists p'. (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s, q) \Rightarrow p'$
by *blast*
}
ultimately show *?thesis*
by *blast*
qed

lemma *ctyping2-term*:

$\llbracket (U, v) \models c (\subseteq A, X) = \text{Some } (B, Y); \exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \rrbracket \Longrightarrow$
 $\exists p'. (c, p) \Rightarrow p'$

proof (*induction (U, v) c A X arbitrary: B Y U v p rule: ctyping2.induct, blast*)

fix $A \ X \ B \ Y \ U \ v \ c_1 \ c_2 \ p$

show

$\llbracket \bigwedge B \ Y \ p. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \Longrightarrow \exists p'. (c_1, p) \Rightarrow p';$
 $\bigwedge q \ B \ Y \ B' \ Y' \ p. (U, v) \models c_1 (\subseteq A, X) = \text{Some } q \Longrightarrow (B, Y) = q \Longrightarrow$
 $(U, v) \models c_2 (\subseteq B, Y) = \text{Some } (B', Y') \Longrightarrow$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \Longrightarrow \exists p'. (c_2, p) \Rightarrow p';$
 $(U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \rrbracket \Longrightarrow$
 $\exists p'. (c_1;; c_2, p) \Rightarrow p'$

by (*rule ctyping2-term-seq*)

next

fix $A \ X \ B \ Y \ U \ v \ c_1 \ c_2 \ p$

show

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

$$\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_1, p) \Rightarrow p';$$

$$(U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B, Y);$$

$$\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \rrbracket \implies$$

$$\exists p'. (c_1 \text{ OR } c_2, p) \Rightarrow p'$$
by (*rule ctyping2-term-or*)

next

fix $A X B Y U v b c_1 c_2 p$

show

$$\llbracket \bigwedge U' q B_1 B_2 B Y p.$$

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

$$(B_1, B_2) = q \implies (U', v) \models c_1 (\subseteq B_1, X) = \text{Some } (B, Y) \implies$$

$$\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_1, p) \Rightarrow p';$$

$$\bigwedge U' q B_1 B_2 B Y p.$$

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

$$(B_1, B_2) = q \implies (U', v) \models c_2 (\subseteq B_2, X) = \text{Some } (B, Y) \implies$$

$$\exists (C, Z) \in U'. \neg C: Z \rightsquigarrow \text{UNIV} \implies \exists p'. (c_2, p) \Rightarrow p';$$

$$(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y);$$

$$\exists (C, Z) \in U. \neg C: Z \rightsquigarrow \text{UNIV} \rrbracket \implies$$

$$\exists p'. (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, p) \Rightarrow p'$$
by (*rule ctyping2-term-if*)

qed (*fastforce split: if-split-asm prod.split-asm*)**+**

lemma *ctyping2-confine-seq*:

assumes

$A: \bigwedge s' f' vs' ws' A B X Y U v. p = (s', f', vs', ws') \implies$
 $(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies \exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $s = s' (\subseteq S) \wedge$
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ vs}'. \text{fst } p \in S] = [] \wedge$
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ ws}'. \text{fst } p \in S] = []$
 $(\text{is } \bigwedge s' - vs' ws' \text{ - - - - -} . \text{ -} \implies \text{ -} \implies \text{ -} \implies$
 $?P s s' vs vs' ws ws')$

assumes

$B: \bigwedge s' f' vs' ws' A B X Y U v. p = (s', f', vs', ws') \implies$
 $(U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies \exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $?P s' s'' vs' vs'' ws' ws'' \text{ and}$
 $C: (c_1, s, f, vs, ws) \Rightarrow p \text{ and}$
 $D: (c_2, p) \Rightarrow (s'', f'', vs'', ws'') \text{ and}$
 $E: (U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (B', Y') \text{ and}$
 $F: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

shows $?P s s'' vs vs'' ws ws''$

proof –

obtain s' **and** f' **and** vs' **and** ws' **where** $G: p = (s', f', vs', ws')$

by (*cases p*)

have $H: (c_1, s, f, vs, ws) \rightarrow^* (\text{SKIP}, s', f', vs', ws')$

using C **and** G **by** (*simp add: big-iff-small*)

have $I: (c_2, s', f', vs', ws') \rightarrow^* (\text{SKIP}, s'', f'', vs'', ws'')$

using D **and** G **by** (*simp add: big-iff-small*)

have $J: vs' = vs @ drop (length vs) vs'$
using H **by** $(rule\ small-steps-in-flow)$
have $vs'' = vs' @ drop (length vs') vs''$
using I **by** $(rule\ small-steps-in-flow)$
hence $K: vs'' = vs @ drop (length vs) vs' @ drop (length vs') vs''$
by $(subst\ (asm)\ J,\ simp)$
have $L: ws' = ws @ drop (length ws) ws'$
using H **by** $(rule\ small-steps-out-flow)$
have $ws'' = ws' @ drop (length ws') ws''$
using I **by** $(rule\ small-steps-out-flow)$
hence $M: ws'' = ws @ drop (length ws) ws' @ drop (length ws') ws''$
by $(subst\ (asm)\ L,\ simp)$
obtain B **and** Y **where**
 $N: (U, v) \models c_1 (\subseteq A, X) = Some (B, Y)$ **and**
 $O: (U, v) \models c_2 (\subseteq B, Y) = Some (B', Y')$
using E **by** $(auto\ split: option.split-asm)$
from $A [OF\ G\ N\ F]$ **and** $B [OF\ G\ O\ F]$ **show** $?thesis$
by $(subst\ K,\ subst\ M,\ simp)$
qed

lemma *ctyping2-confine-or-lhs*:

assumes

$A: \bigwedge A\ B\ X\ Y\ U\ v. (U, v) \models c_1 (\subseteq A, X) = Some (B, Y) \implies$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $s = s' (\subseteq S) \wedge$
 $[p \leftarrow drop (length vs) vs'. fst\ p \in S] = [] \wedge$
 $[p \leftarrow drop (length ws) ws'. fst\ p \in S] = []$
(is $\bigwedge - \dots - \implies - \implies ?P$

assumes

$B: (U, v) \models c_1\ OR\ c_2 (\subseteq A, X) = Some (B', Y')$ **and**
 $C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

shows $?P$

proof –

obtain B **and** Y **where** $(U, v) \models c_1 (\subseteq A, X) = Some (B, Y)$
using B **by** $(auto\ split: option.split-asm)$
with A **and** C **show** $?thesis$
by $simp$

qed

lemma *ctyping2-confine-or-rhs*:

assumes

$A: \bigwedge A\ B\ X\ Y\ U\ v. (U, v) \models c_2 (\subseteq A, X) = Some (B, Y) \implies$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $s = s' (\subseteq S) \wedge$
 $[p \leftarrow drop (length vs) vs'. fst\ p \in S] = [] \wedge$
 $[p \leftarrow drop (length ws) ws'. fst\ p \in S] = []$
(is $\bigwedge - \dots - \implies - \implies ?P$

assumes

$B: (U, v) \models c_1\ OR\ c_2 (\subseteq A, X) = Some (B', Y')$ **and**

$C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$
shows $?P$
proof –
obtain B and Y **where** $(U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y)$
using B **by** $(\text{auto split: option.split-asm})$
with A and C **show** $?thesis$
by simp
qed

lemma *ctyping2-confine-if-true:*

assumes

$A: \bigwedge A B X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $s = s' (\subseteq S) \wedge$
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ } vs'. \text{fst } p \in S] = [] \wedge$
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ } ws'. \text{fst } p \in S] = []$
(is $\bigwedge - \dots - \implies - \implies ?P$)

assumes

$B: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y)$ **and**
 $C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

shows $?P$

proof –

obtain B_1 and B_1' and Y_1 **where**

$(\text{insert } (\text{Univ? } A \ X, \text{ bvars } b) \ U, v) \models c_1 (\subseteq B_1, X) = \text{Some } (B_1', Y_1)$
using B **by** $(\text{auto split: option.split-asm prod.split-asm})$

with A and C **show** $?thesis$

by simp

qed

lemma *ctyping2-confine-if-false:*

assumes

$A: \bigwedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies$
 $s = s' (\subseteq S) \wedge$
 $[p \leftarrow \text{drop } (\text{length } vs) \text{ } vs'. \text{fst } p \in S] = [] \wedge$
 $[p \leftarrow \text{drop } (\text{length } ws) \text{ } ws'. \text{fst } p \in S] = []$
(is $\bigwedge - \dots - \implies - \implies ?P$)

assumes

$B: (U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y)$ **and**
 $C: \exists (C, Z) \in U. C: Z \rightsquigarrow | S$

shows $?P$

proof –

obtain B_2 and B_2' and Y_2 **where**

$(\text{insert } (\text{Univ? } A \ X, \text{ bvars } b) \ U, v) \models c_2 (\subseteq B_2, X) = \text{Some } (B_2', Y_2)$
using B **by** $(\text{auto split: option.split-asm prod.split-asm})$

with A and C **show** $?thesis$

by simp

qed

lemma *ctyping2-confine*:

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

proof (*induction* $(c, s, f, vs, ws) (s', f', vs', ws')$ *arbitrary*:
 $c s f vs ws s' f' vs' ws' A B X Y U v$ *rule*: *big-step.induct*)

fix $A B X Y U v c_1 c_2 p s f vs ws s' f' vs' ws'$

show

$\llbracket \wedge s' f' vs' ws' A B X Y U v. p = (s', f', vs', ws') \rrbracket \Longrightarrow$
 $(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \Longrightarrow ?P s s' vs vs' ws ws';$
 $\wedge s f vs ws A B X Y U v. p = (s, f, vs, ws) \Longrightarrow$
 $(U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \Longrightarrow$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \Longrightarrow ?P s s' vs vs' ws ws';$
 $(c_1, s, f, vs, ws) \Rightarrow p;$
 $(c_2, p) \Rightarrow (s', f', vs', ws');$
 $(U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \rrbracket \Longrightarrow$
 $?P s s' vs vs' ws ws'$

by (*rule ctyping2-confine-seq*)

next

fix $A B X Y U v c_1 c_2 s vs ws s' vs' ws'$

show

$\llbracket \wedge A B X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \rrbracket \Longrightarrow$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \Longrightarrow ?P s s' vs vs' ws ws';$
 $(U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \rrbracket \Longrightarrow$
 $?P s s' vs vs' ws ws'$

by (*rule ctyping2-confine-or-lhs*)

next

fix $A B X Y U v c_1 c_2 s vs ws s' vs' ws'$

show

$\llbracket \wedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \rrbracket \Longrightarrow$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \Longrightarrow ?P s s' vs vs' ws ws';$
 $(U, v) \models c_1 \text{ OR } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \rrbracket \Longrightarrow$
 $?P s s' vs vs' ws ws'$

by (*rule ctyping2-confine-or-rhs*)

next

fix $A B X Y U v b c_1 c_2 s vs ws s' vs' ws'$

show

$\llbracket \wedge A B X Y U v. (U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \rrbracket \Longrightarrow$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \Longrightarrow ?P s s' vs vs' ws ws';$
 $(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \rrbracket \Longrightarrow$
 $?P s s' vs vs' ws ws'$

by (rule *ctyping2-confine-if-true*)
next
 fix $A B X Y U v b c_1 c_2 s vs ws s' vs' ws'$
show
 $\llbracket \bigwedge A B X Y U v. (U, v) \models c_2 (\subseteq A, X) = \text{Some } (B, Y) \implies$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \implies ?P s s' vs vs' ws ws';$
 $(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 (\subseteq A, X) = \text{Some } (B, Y);$
 $\exists (C, Z) \in U. C: Z \rightsquigarrow | S \rrbracket \implies$
 $?P s s' vs vs' ws ws'$
 by (rule *ctyping2-confine-if-false*)
qed (force *split: if-split-asm prod.split-asm*)+

lemma *eq-states-assign*:

assumes
 $A: S \subseteq \{y. s = t (\subseteq \text{sources } [x ::= a] \text{ vs } s f y)\}$ **and**
 $B: x \in S$ **and**
 $C: s \in \text{Univ } A (\subseteq \text{state} \cap X)$ **and**
 $D: \text{Univ? } A X: \text{avars } a \rightsquigarrow \{x\}$
shows $s = t (\subseteq \text{avars } a)$
proof –
obtain r **where** $E: r \in A$ **and** $F: s = r (\subseteq \text{state} \cap X)$
using C **by** *blast*
have $\text{avars } a \subseteq \{y. s: \text{dom } y \rightsquigarrow \text{dom } x\}$
proof (*cases state* $\subseteq X$)
case *True*
with F **have** $\text{interf } s = \text{interf } r$
by (*blast intro: interf-state*)
with D **and** E **show** *?thesis*
by (*auto simp: univ-states-if-def split: if-split-asm*)
next
case *False*
with D **and** E **show** *?thesis*
by (*auto simp: univ-states-if-def split: if-split-asm*)
qed
moreover **have** $s = t (\subseteq \text{sources } [x ::= a] \text{ vs } s f x)$
using A **and** B **by** *blast*
hence $s = t (\subseteq \{y. s: \text{dom } y \rightsquigarrow \text{dom } x \wedge y \in \text{avars } a\})$
by (*subst (asm) append-Nil [symmetric]*),
simp only: sources.simps, auto)
ultimately show *?thesis*
by *blast*
qed

lemma *eq-states-while*:

assumes
 $A: S \subseteq \{x. s = t (\subseteq \text{sources-aux } ((\text{bvars } b) \# \text{cs}) \text{ vs } s f x)\}$ **and**
 $B: S \neq \{\}$ **and**
 $C: s \in \text{Univ } A (\subseteq \text{state} \cap X) \cup \text{Univ } C (\subseteq \text{state} \cap Y)$ **and**

$D: Univ? A X \cup Univ? C Y: bvars b \rightsquigarrow UNIV$
shows $s = t (\subseteq bvars b)$
proof –
from C **have** $\{s\}: bvars b \rightsquigarrow UNIV$
proof
assume $s \in Univ A (\subseteq state \cap X)$
then obtain r **where** $E: r \in A$ **and** $F: s = r (\subseteq state \cap X)$
by *blast*
show *?thesis*
proof (*cases state $\subseteq X$*)
case *True*
with F **have** *interf s = interf r*
by (*blast intro: interf-state*)
with D **and** E **show** *?thesis*
by (*auto simp: univ-states-if-def split: if-split-asm*)
qed (*insert D E, auto simp: univ-states-if-def split: if-split-asm*)
next
assume $s \in Univ C (\subseteq state \cap Y)$
then obtain r **where** $E: r \in C$ **and** $F: s = r (\subseteq state \cap Y)$
by *blast*
show *?thesis*
proof (*cases state $\subseteq Y$*)
case *True*
with F **have** *interf s = interf r*
by (*blast intro: interf-state*)
with D **and** E **show** *?thesis*
by (*auto simp: univ-states-if-def split: if-split-asm*)
qed (*insert D E, auto simp: univ-states-if-def split: if-split-asm*)
qed
hence $\forall x. bvars b \subseteq sources\text{-aux} (\langle bvars b \rangle \# cs) vs s f x$
by (*blast intro!: sources-ax-observe-hd*)
thus *?thesis*
using A **and** B **by** *blast*
qed

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 (*erule UnE*)
assume $H: s \in Univ A (\subseteq state \cap X)$
have $s \in Univ B_1 (\subseteq state \cap X)$

```

    using G by (insert btyping2-approx [OF B H], simp)
  with A and E have s' ∈ Univ D (⊆ state ∩ Z)
    by (rule ctyping2-approx)
  moreover have D ⊆ C ∧ Y ⊆ Z
    using C and E by (rule ctyping1-ctyping2)
  ultimately show ?thesis
    by blast
next
assume H: s ∈ Univ C (⊆ state ∩ Y)
have s ∈ Univ B1' (⊆ state ∩ Y)
  using G by (insert btyping2-approx [OF D H], simp)
with A and F have s' ∈ Univ D' (⊆ state ∩ Z')
  by (rule ctyping2-approx)
moreover obtain C' and Y' where I: ⊢ c (⊆ B1', Y) = (C', Y')
  by (cases ⊢ c (⊆ B1', Y), simp)
hence D' ⊆ C' ∧ Y' ⊆ Z'
  using F by (rule ctyping1-ctyping2)
ultimately have s' ∈ Univ C' (⊆ state ∩ Y')
  by blast
moreover have J: ⊢ c (⊆ C, Y) = (C, Y)
  using C by (rule ctyping1-idem)
have B1' ⊆ C
  using D by (blast dest: btyping2-un-eq)
with J and I have C' ⊆ C ∧ Y' ⊆ Y'
  by (rule ctyping1-mono, simp)
ultimately show ?thesis
  by blast
qed

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!*]:

$$\llbracket (SKIP, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1);$$

$$(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2) \rrbracket \implies$$

$$ok\text{-flow-aux } U \ c_1 \ c_2 \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ (\text{flow } cfs_2)$$
by (*fastforce dest: small-stepsl-skip*)

lemma *ctyping2-correct-aux-assign*:

assumes

$A: (U, v) \models x ::= a \ (\subseteq A, X) = \text{Some } (C, Y)$ **and**

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

$C: (x ::= a, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$ **and**

$D: (c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

shows *ok-flow-aux* $U \ c_1 \ c_2 \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ (\text{flow } cfs_2)$

proof –

from A **have** $E: \forall (B, Y) \in \text{insert } (\text{Univ? } A \ X, \text{avars } a) \ U. B: Y \rightsquigarrow \{x\}$

by (*simp split: if-split-asm*)

have

$(c_1, s_1, f, vs_1, ws_1) = (x ::= a, s, f, vs_0, ws_0) \vee$

$(c_1, s_1, f, vs_1, ws_1) = (SKIP, s(x := \text{aval } a \ s), f, vs_0, ws_0)$

(is $?P \vee ?Q$)

using C **by** (*blast dest: small-stepsl-assign*)

thus *?thesis*

proof

assume $?P$

hence $(x ::= a, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

using D **by** *simp*

hence

$(c_2, s_2, f, vs_2, ws_2) = (x ::= a, s, f, vs_0, ws_0) \wedge$

$\text{flow } cfs_2 = [] \vee$

$(c_2, s_2, f, vs_2, ws_2) = (SKIP, s(x := \text{aval } a \ s), f, vs_0, ws_0) \wedge$

$\text{flow } cfs_2 = [x ::= a]$

(is $?P' \vee -$)

by (*rule small-stepsl-assign*)

thus *?thesis*

proof (*rule disjE, erule-tac [2] conjE*)

assume $?P'$

with $\langle ?P \rangle$ **show** *?thesis*

by *fastforce*

next

assume

$F: (c_2, s_2, f, vs_2, ws_2) = (SKIP, s(x := aval a s), f, vs_0, ws_0)$ **and**
 $G: flow\ cfs_2 = [x ::= a]$
 (is $?cs = -$)
show *?thesis*
proof (rule *conjI*, clarify)
 fix $t_1\ f'\ vs_1'\ ws_1'$
 let $?t_2 = t_1(x := aval a t_1)$
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
ok-flow-aux-1 $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
ok-flow-aux-2 $s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (rule *exI* [of - *SKIP*], rule *exI* [of - $?t_2$],
 rule *exI* [of - vs_1], rule *exI* [of - ws_1])
 {
 fix S
 assume $S \subseteq \{y. s = t_1 (\subseteq sources\ [x ::= a]\ vs_0\ s\ f\ y)\}$ **and**
 $x \in S$
 hence $s = t_1 (\subseteq avars\ a)$
 using B by (rule *eq-states-assign*, insert E , *simp*)
 hence $aval\ a\ s = aval\ a\ t_1$
 by (rule *avars-aval*)
 }
moreover {
 fix $S\ y$
 assume $S \subseteq \{y. s = t_1 (\subseteq sources\ [x ::= a]\ vs_0\ s\ f\ y)\}$ **and**
 $y \in S$
 hence $s = t_1 (\subseteq sources\ [x ::= a]\ vs_0\ s\ f\ y)$
 by *blast*
moreover assume $y \neq x$
ultimately have $s\ y = t_1\ y$
 by (*subst* (*asm*) *append-Nil* [*symmetric*],
simp only: *sources.simps*, *simp*)
 }
ultimately show
ok-flow-aux-1 $c_1\ c_2\ SKIP\ s_1\ t_1\ ?t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_1'\ ws_1'\ ws_1'\ ?cs \wedge$
ok-flow-aux-2 $s_1\ s_2\ t_1\ ?t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_1'\ ?cs$
 using F and G and $\langle ?P \rangle$ by *auto*
qed
 qed (*insert* $E\ G$, *fastforce*)
qed
next
 assume $?Q$
moreover from *this* have
 $(c_2, s_2, f, vs_2, ws_2) = (SKIP, s_1, f, vs_1, ws_1) \wedge flow\ cfs_2 = []$
 using D by (*blast* *intro!*: *small-steps1-skip*)
ultimately show *?thesis*

by *fastforce*
qed
qed

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 –

from A **have** $D: \forall (B, Y) \in U. B: Y \rightsquigarrow \{x\}$

by (*simp split: if-split-asm*)

let $?n = length\ [p \leftarrow vs_0.\ fst\ p = x]$

have

$(c_1, s_1, f, vs_1, ws_1) = (IN\ x, s, f, vs_0, ws_0) \vee$

$(c_1, s_1, f, vs_1, ws_1) =$

$(SKIP, s(x := f\ x\ ?n), f, vs_0 @ [(x, f\ x\ ?n)], ws_0)$

(**is** $?P \vee ?Q$)

using B **by** (*auto dest: small-steps1-input simp: Let-def*)

thus *?thesis*

proof

assume $?P$

hence $(IN\ x, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

using C **by** *simp*

hence

$(c_2, s_2, f, vs_2, ws_2) = (IN\ x, s, f, vs_0, ws_0) \wedge$

$flow\ cfs_2 = [] \vee$

$(c_2, s_2, f, vs_2, ws_2) =$

$(SKIP, s(x := f\ x\ ?n), f, vs_0 @ [(x, f\ x\ ?n)], ws_0) \wedge$

$flow\ cfs_2 = [IN\ x]$

(**is** $?P' \vee -$)

by (*auto dest: small-steps1-input simp: Let-def*)

thus *?thesis*

proof (*rule disjE, erule-tac [2] conjE*)

assume $?P'$

with $\langle ?P \rangle$ **show** *?thesis*

by *fastforce*

next

assume

$E: (c_2, s_2, f, vs_2, ws_2) =$

$(SKIP, s(x := f\ x\ ?n), f, vs_0 @ [(x, f\ x\ ?n)], ws_0)$ **and**

$F: flow\ cfs_2 = [IN\ x]$

(**is** $?cs = -$)

show *?thesis*

proof (*rule conjI, clarify*)

fix $t_1\ f'\ vs_1'\ ws_1'$

let $?n' = length\ [p \leftarrow vs_1' :: inputs.\ fst\ p = x]$

let $?t_2 = t_1(x := f'\ x\ ?n') :: state$ **and**

$?vs_2' = vs_1' @ [(x, f' x ?n')$
show $\exists c_2' t_2 vs_2' ws_2'$.
ok-flow-aux-1 $c_1 c_2 c_2' s_1 t_1 t_2 ff'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_2 ff' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 ff' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
proof (*rule exI [of - SKIP]*, *rule exI [of - ?t2]*,
rule exI [of - ?vs2], *rule exI [of - ws1]*)
{
 fix S
 assume $f = f' (\subseteq vs_0, vs_1',$
 $\bigcup \{tags [IN x] vs_0 s f y \mid y. y \in S\})$
 (**is** $- = - (\subseteq -, -, ?T)$)
 moreover assume $x \in S$
 hence $tags [IN x] vs_0 s f x \subseteq ?T$
 by *blast*
 ultimately have $f = f' (\subseteq vs_0, vs_1', tags [IN x] vs_0 s f x)$
 by (*rule eq-streams-subset*)
 moreover have $tags [IN x] vs_0 s f x = \{(x, 0)\}$
 by (*subst append-Nil [symmetric]*,
 simp only: tags.simps, simp)
 ultimately have $f x (length [p \leftarrow vs_0. fst p = x]) =$
 $f' x (length [p \leftarrow vs_1'. fst p = x])$
 by (*simp add: eq-streams-def*)
}
moreover
{
 fix $S y$
 assume $S \subseteq \{y. s = t_1 (\subseteq sources [IN x] vs_0 s f y)\}$ **and**
 $y \in S$
 hence $s = t_1 (\subseteq sources [IN x] vs_0 s f y)$
 by *blast*
 moreover assume $y \neq x$
 ultimately have $s y = t_1 y$
 by (*subst (asm) append-Nil [symmetric]*,
 simp only: sources.simps, simp)
}
ultimately show
ok-flow-aux-1 $c_1 c_2 SKIP s_1 t_1 ?t_2 ff'$
 $vs_1 vs_1' vs_2 ?vs_2' ws_1' ws_1' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 ?t_2 ff' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 ff' vs_1 vs_1' ws_1 ws_1' ws_2 ws_1' ?cs$
using E **and** F **and** $\langle ?P \rangle$ **by** *auto*
qed
qed (*insert D F, fastforce*)
qed
next
assume $?Q$
moreover from *this* **have**

$(c_2, s_2, f, vs_2, ws_2) = (SKIP, s_1, f, vs_1, ws_1) \wedge \text{flow } cfs_2 = []$
using C **by** (*blast intro!: small-stepsl-skip*)
ultimately show *?thesis*
by *fastforce*
qed
qed

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 –

from A **have** $D: \forall (B, Y) \in U. B: Y \rightsquigarrow \{x\}$

by (*simp split: if-split-asm*)

have

$(c_1, s_1, f, vs_1, ws_1) = (OUT\ x, s, f, vs_0, ws_0) \vee$

$(c_1, s_1, f, vs_1, ws_1) = (SKIP, s, f, vs_0, ws_0 @ [(x, s\ x)])$

(**is** $?P \vee ?Q$)

using B **by** (*blast dest: small-stepsl-output*)

thus *?thesis*

proof

assume $?P$

hence $(OUT\ x, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

using C **by** *simp*

hence

$(c_2, s_2, f, vs_2, ws_2) = (OUT\ x, s, f, vs_0, ws_0) \wedge$

$\text{flow } cfs_2 = [] \vee$

$(c_2, s_2, f, vs_2, ws_2) = (SKIP, s, f, vs_0, ws_0 @ [(x, s\ x)]) \wedge$

$\text{flow } cfs_2 = [OUT\ x]$

(**is** $?P' \vee -$)

by (*rule small-stepsl-output*)

thus *?thesis*

proof (*rule disjE, erule-tac [2] conjE*)

assume $?P'$

with $\langle ?P \rangle$ **show** *?thesis*

by *fastforce*

next

assume

$E: (c_2, s_2, f, vs_2, ws_2) = (SKIP, s, f, vs_0, ws_0 @ [(x, s\ x)])$ **and**

$F: \text{flow } cfs_2 = [OUT\ x]$

(**is** $?cs = -$)

show *?thesis*

proof (*rule conjI, clarify*)

fix $t_1\ f'\ vs_1'\ ws_1'$

let $?ws_2' = ws_1' @ [(x, t_1\ x)] :: \text{outputs}$

show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.

ok-flow-aux-1 $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'$

$vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \wedge$
 $ok\text{-flow-aux-2} \ s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow-aux-3} \ s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$

proof (*rule exI [of - SKIP]*, *rule exI [of - t₁]*,
rule exI [of - vs₁'], *rule exI [of - ?ws₂']*)

{
 fix $S \ y$
 assume $S \subseteq \{y. s = t_1 (\subseteq \text{sources [OUT } x] \ vs_0 \ s \ f \ y))\}$ **and**
 $y \in S$
 hence $s = t_1 (\subseteq \text{sources [OUT } x] \ vs_0 \ s \ f \ y)$
 by *blast*
 hence $s \ y = t_1 \ y$
 by (*subst (asm) append-Nil [symmetric]*,
 simp only: sources.simps, simp)
 }

moreover {
 fix S
 assume $S \subseteq \{y. s = t_1 (\subseteq \text{sources-out [OUT } x] \ vs_0 \ s \ f \ y))\}$ **and**
 $x \in S$
 hence $s = t_1 (\subseteq \text{sources-out [OUT } x] \ vs_0 \ s \ f \ x)$
 by *blast*
 hence $s \ x = t_1 \ x$
 by (*subst (asm) append-Nil [symmetric]*,
 simp only: sources-out.simps, simp)
 }

ultimately show
 $ok\text{-flow-aux-1} \ c_1 \ c_2 \ SKIP \ s_1 \ t_1 \ t_1 \ f \ f'$
 $vs_1 \ vs_1' \ vs_2 \ vs_1' \ ws_1' \ ?ws_2' \ ?cs \wedge$
 $ok\text{-flow-aux-2} \ s_1 \ s_2 \ t_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow-aux-3} \ s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ?ws_2' \ ?cs$
using E **and** F **and** $\langle ?P \rangle$ **by** *auto*

qed
qed (*insert D F, fastforce*)
qed

next
assume $?Q$
moreover from this have
 $(c_2, s_2, f, vs_2, ws_2) = (SKIP, s_1, f, vs_1, ws_1) \wedge \text{flow } cfs_2 = []$
using C **by** (*blast intro!: small-steps1-skip*)
ultimately show $?thesis$
by *fastforce*

qed
qed

lemma *ctyping2-correct-aux-seq*:
assumes
 $A: (U, v) \models c_1;; c_2 (\subseteq A, X) = \text{Some } (C, Z)$ **and**
 $B: \bigwedge B \ Y \ c' \ c'' \ s \ s_1 \ s_2 \ vs_0 \ vs_1 \ vs_2 \ ws_0 \ ws_1 \ ws_2 \ cfs_1 \ cfs_2.$
 $(U, v) \models c_1 (\subseteq A, X) = \text{Some } (B, Y) \implies$

$s \in Univ A (\subseteq state \cap X) \implies$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$
ok-flow-aux $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$ (*flow cfs₂*) **and**
C: $\bigwedge p B Y C Z c' c'' s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$
 $(U, v) \models c_1 (\subseteq A, X) = Some p \implies (B, Y) = p \implies$
 $(U, v) \models c_2 (\subseteq B, Y) = Some (C, Z) \implies$
 $s \in Univ B (\subseteq state \cap Y) \implies$
 $(c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$
ok-flow-aux $U c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$ (*flow cfs₂*) **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 cfs₂*)

proof –

from A **obtain** B **and** Y **where**

$G: (U, v) \models c_1 (\subseteq A, X) = Some (B, Y)$ **and**

$H: (U, v) \models c_2 (\subseteq B, Y) = Some (C, Z)$

by (*auto split: option.split-asm*)

have

$(\exists c cfs. c' = c;; c_2 \wedge$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c, s_1, f, vs_1, ws_1)) \vee$
 $(\exists s' p cfs cfs'.$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (SKIP, s', p) \wedge$
 $(c_2, s', p) \rightarrow^*\{cfs'\} (c', s_1, f, vs_1, ws_1))$

using E **by** (*fastforce dest: small-stepsl-seq*)

thus *?thesis*

proof (*rule disjE, (erule-tac exE)+, (erule-tac [2] exE)+,*
erule-tac [!] conjE)

fix $c_1' cfs$

assume

$I: c' = c_1';; c_2$ **and**

$J: (c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c_1', s_1, f, vs_1, ws_1)$

hence $(c_1';; c_2, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2)$

using F **by** *simp*

hence

$(\exists d cfs'. c'' = d;; c_2 \wedge$
 $(c_1', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (d, s_2, f, vs_2, ws_2) \wedge$
 $flow cfs_2 = flow cfs') \vee$
 $(\exists p cfs' cfs''.$
 $(c_1', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (SKIP, p) \wedge$
 $(c_2, p) \rightarrow^*\{cfs''\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow cfs_2 = flow cfs' @ flow cfs'')$

by (*blast dest: small-stepsl-seq*)

thus *?thesis*

proof (*rule disjE, (erule-tac exE)+, (erule-tac [2] exE)+,*
(erule-tac [!] conjE)+)

fix $c_1'' cfs'$

assume
 $K: c'' = c_1'';; c_2$ **and**
 $L: (c_1', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (c_1'', s_2, f, vs_2, ws_2)$ **and**
 $M: \text{flow } cfs_2 = \text{flow } cfs'$
 (is $?cs = ?cs'$)
have $N: \text{ok-flow-aux } U \ c_1' \ c_1'' \ s_1 \ s_2 \ f \ vs_1 \ vs_2 \ ws_1 \ ws_2 \ ?cs'$
using $B [OF \ G \ D \ J \ L]$.
show $?thesis$
proof (rule $conjI$, clarify)
fix $t_1 \ f' \ vs_1' \ ws_1'$
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $ok\text{-flow-aux-1 } c_1' \ c_1'' \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f'$
 $vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \wedge$
 $ok\text{-flow-aux-2 } s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow-aux-3 } s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$
 (is $?P1 \wedge ?P2 \wedge ?P3$)
using M **and** N **by** $fastforce$
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** $auto$
show $\exists c_2' \ t_2 \ vs_2' \ ws_2'$.
 $ok\text{-flow-aux-1 } c' \ c'' \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f'$
 $vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \wedge$
 $ok\text{-flow-aux-2 } s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow-aux-3 } s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$
proof (rule $exI [of - c_2';; c_2]$, rule $exI [of - t_2]$,
 rule $exI [of - vs_2']$, rule $exI [of - ws_2']$)
 {
fix S
assume $S \neq \{\}$ **and**
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs \ vs_1 \ s_1 \ f \ x)\}$ **and**
 $f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags-aux } ?cs \ vs_1 \ s_1 \ f \ x \mid x. x \in S\})$
hence
 $(c', t_1, f', vs_1', ws_1') \rightarrow^* (c_2';; c_2, t_2, f', vs_2', ws_2') \wedge$
 $\text{map } fst [p \leftarrow \text{drop } (\text{length } vs_1) \ vs_2. \text{fst } p \in S] =$
 $\text{map } fst [p \leftarrow \text{drop } (\text{length } vs_1') \ vs_2'. \text{fst } p \in S]$
using I **and** $\langle ?P1 \rangle$ **by** (blast intro: star-seq2)
 }
thus
 $ok\text{-flow-aux-1 } c' \ c'' \ (c_2';; c_2) \ s_1 \ t_1 \ t_2 \ f \ f'$
 $vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \wedge$
 $ok\text{-flow-aux-2 } s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow-aux-3 } s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$
using K **and** $\langle ?P2 \rangle$ **and** $\langle ?P3 \rangle$ **by** $simp$
qed
 (simp add: $M \ N$)
next
fix $p \ cfs' \ cfs''$
assume $(c_1', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (SKIP, p)$
moreover from this obtain s_1' **and** vs **and** ws **where**
 $K: p = (s_1', f, vs, ws)$

by (*blast dest: small-stepsl-stream*)
ultimately have
 $L: (c_1', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (SKIP, s_1', f, vs, ws)$
 by *simp*
assume $(c_2, p) \rightarrow^*\{cfs''\} (c'', s_2, f, vs_2, ws_2)$
with K have
 $M: (c_2, s_1', f, vs, ws) \rightarrow^*\{cfs''\} (c'', s_2, f, vs_2, ws_2)$
 by *simp*
assume N : $flow\ cfs_2 = flow\ cfs' @ flow\ cfs''$
 (*is* $(?cs :: flow) = ?cs' @ ?cs''$)
have O : *ok-flow-aux* $U\ c_1'\ SKIP\ s_1\ s_1'\ f\ vs_1\ vs\ ws_1\ ws\ ?cs'$
 using $B [OF\ G\ D\ J\ L]$.
have $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs @ cfs'\} (SKIP, s_1', f, vs, ws)$
 using J and L by (*simp add: small-stepsl-append*)
hence $(c_1, s, f, vs_0, ws_0) \Rightarrow (s_1', f, vs, ws)$
 by (*auto dest: small-stepsl-steps simp: big-iff-small*)
hence P : $s_1' \in Univ\ B (\subseteq state \cap Y)$
 using G and D by (*rule ctying2-approx*)
have Q : *ok-flow-aux* $U\ c_2\ c''\ s_1'\ s_2\ f\ vs\ vs_2\ ws\ ws_2\ ?cs''$
 using $C [OF\ G - H\ P - M, of\ vs\ ws\ \square]$ by *simp*
show *?thesis*
proof (*rule conjI, clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_1'' and t_1' and vs_1'' and ws_1'' where
ok-flow-aux-1 $c_1'\ SKIP\ c_1''\ s_1\ t_1\ t_1'\ f\ f'$
 $vs_1\ vs_1'\ vs\ vs_1''\ ws_1'\ ws_1''\ ?cs' \wedge$
ok-flow-aux-2 $s_1\ s_1'\ t_1\ t_1'\ f\ f'\ vs_1\ vs_1'\ ?cs' \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws\ ws_1''\ ?cs'$
 (*is* $- \wedge ?P2 \wedge ?P3$)
 using O by *fastforce*
hence
ok-flow-aux-1 $c_1'\ SKIP\ SKIP\ s_1\ t_1\ t_1'\ f\ f'$
 $vs_1\ vs_1'\ vs\ vs_1''\ ws_1'\ ws_1''\ ?cs'$
 (*is* $?P1$) and $?P2$ and $?P3$ by *auto*
obtain c_2' and t_2 and vs_2' and ws_2' where
ok-flow-aux-1 $c_2\ c''\ c_2'\ s_1'\ t_1'\ t_2\ f\ f'$
 $vs\ vs_1''\ vs_2\ vs_2'\ ws_1''\ ws_2'\ ?cs'' \wedge$
ok-flow-aux-2 $s_1'\ s_2\ t_1'\ t_2\ f\ f'\ vs\ vs_1''\ ?cs'' \wedge$
ok-flow-aux-3 $s_1'\ t_1'\ f\ f'\ vs\ vs_1''\ ws\ ws_1''\ ws_2\ ws_2'\ ?cs''$
 (*is* $?P1' \wedge ?P2' \wedge ?P3'$)
 using Q by *fastforce*
hence $?P1'$ and $?P2'$ and $?P3'$ by *auto*
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
ok-flow-aux-1 $c'\ c''\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
ok-flow-aux-2 $s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (*rule exI [of - c_2']*, *rule exI [of - t_2]*,
rule exI [of - vs_2'], *rule exI [of - ws_2']*)

```

{
  fix S
  assume
    R: S ≠ {} and
    S: S ⊆ {x. s1 = t1 (⊆ sources-aux (?cs' @ ?cs'') vs1 s1 f x)} and
    T: f = f' (⊆ vs1, vs1',
      ⋃ {tags-aux (?cs' @ ?cs'') vs1 s1 f x | x. x ∈ S})
      (is - = - (⊆ -, -, ?T))
  have ∀ x. sources-aux ?cs' vs1 s1 f x ⊆
    sources-aux (?cs' @ ?cs'') vs1 s1 f x
    by (blast intro: subsetD [OF sources-aux-append])
  hence S ⊆ {x. s1 = t1 (⊆ sources-aux ?cs' vs1 s1 f x)}
    using S by blast
  moreover have ⋃ {tags-aux ?cs' vs1 s1 f x | x. x ∈ S} ⊆ ?T
    (is ?T' ⊆ -)
    by (blast intro: subsetD [OF tags-aux-append])
  with T have f = f' (⊆ vs1, vs1', ?T')
    by (rule eq-streams-subset)
  ultimately have
    (c1', t1, f', vs1', ws1') →* (SKIP, t1', f', vs1'', ws1'') ∧
    map fst [p←drop (length vs1) vs. fst p ∈ S] =
    map fst [p←drop (length vs1') vs1''. fst p ∈ S]
    (is ?Q1 ∧ ?Q2)
    using R and ⟨?P1⟩ by simp
  hence ?Q1 and ?Q2 by auto
  have S ⊆ {x. s1' = t1' (⊆ sources-aux ?cs'' vs s1' f x)}
    by (rule sources-aux-rhs [OF S T L ⟨?P2⟩])
  moreover have f = f' (⊆ vs, vs1'',
    ⋃ {tags-aux ?cs'' vs s1' f x | x. x ∈ S})
    by (rule tags-aux-rhs [OF S T L ⟨?Q1⟩ ⟨?P1⟩])
  ultimately have
    (c2, t1', f', vs1'', ws1'') →* (c2', t2, f', vs2', ws2') ∧
    (c'' = SKIP) = (c2' = SKIP) ∧
    map fst [p←drop (length vs) vs2. fst p ∈ S] =
    map fst [p←drop (length vs1'') vs2'. fst p ∈ S]
    (is ?Q1' ∧ ?R2 ∧ ?Q2')
    using R and ⟨?P1'⟩ by simp
  hence ?Q1' and ?R2 and ?Q2' by auto
  from I and ⟨?Q1⟩ and ⟨?Q1'⟩ have
    (c', t1, f', vs1', ws1') →* (c2', t2, f', vs2', ws2')
    (is ?R1)
    by (blast intro: star-seq2 star-trans)
  moreover have
    map fst [p←drop (length vs1) vs2. fst p ∈ S] =
    map fst [p←drop (length vs1') vs2'. fst p ∈ S]
    by (rule small-steps-inputs [OF L M ⟨?Q1⟩ ⟨?Q1'⟩ ⟨?Q2⟩ ⟨?Q2'⟩])
  ultimately have ?R1 ∧ ?R2 ∧ ?this
    using ⟨?R2⟩ by simp
}

```

moreover {
fix S
assume
 $R: S \neq \{\}$ **and**
 $S: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } (?cs' @ ?cs'') vs_1 s_1 f x)\}$ **and**
 $T: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{\text{tags } (?cs' @ ?cs'') vs_1 s_1 f x \mid x. x \in S\})$
 $(\text{is } - = - (\subseteq -, -, ?T))$
have $\forall x. \text{sources-aux } (?cs' @ ?cs'') vs_1 s_1 f x \subseteq$
 $\text{sources } (?cs' @ ?cs'') vs_1 s_1 f x$
by (*blast intro: subsetD [OF sources-aux-sources]*)
moreover have $\forall x. \text{sources-aux } ?cs' vs_1 s_1 f x \subseteq$
 $\text{sources-aux } (?cs' @ ?cs'') vs_1 s_1 f x$
by (*blast intro: subsetD [OF sources-aux-append]*)
ultimately have $U: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs' vs_1 s_1 f x)\}$
using S **by** *blast*
have $\bigcup \{\text{tags-aux } (?cs' @ ?cs'') vs_1 s_1 f x \mid x. x \in S\} \subseteq ?T$
 $(\text{is } ?T' \subseteq -)$
by (*blast intro: subsetD [OF tags-aux-tags]*)
moreover have $\bigcup \{\text{tags-aux } ?cs' vs_1 s_1 f x \mid x. x \in S\} \subseteq ?T'$
 $(\text{is } ?T'' \subseteq -)$
by (*blast intro: subsetD [OF tags-aux-append]*)
ultimately have $?T'' \subseteq ?T$
by *simp*
with T **have** $f = f' (\subseteq vs_1, vs_1', ?T'')$
by (*rule eq-streams-subset*)
hence $V: (c_1', t_1, f', vs_1', ws_1') \rightarrow^* (SKIP, t_1', f', vs_1'', ws_1'')$
using R **and** U **and** $\langle ?P1 \rangle$ **by** *simp*
have $S \subseteq \{x. s_1' = t_1' (\subseteq \text{sources } ?cs'' vs s_1' f x)\}$
by (*rule sources-rhs [OF S T L $\langle ?P2 \rangle$]*)
moreover have $f = f' (\subseteq vs, vs_1'',$
 $\bigcup \{\text{tags } ?cs'' vs s_1' f x \mid x. x \in S\})$
by (*rule tags-rhs [OF S T L V $\langle ?P1 \rangle$]*)
ultimately have $s_2 = t_2 (\subseteq S)$
using $\langle ?P2' \rangle$ **by** *blast*
}
moreover {
fix S
assume
 $R: S \neq \{\}$ **and**
 $S: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } (?cs' @ ?cs'') vs_1 s_1 f x)\}$ **and**
 $T: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{\text{tags-out } (?cs' @ ?cs'') vs_1 s_1 f x \mid x. x \in S\})$
 $(\text{is } - = - (\subseteq -, -, ?T))$
have $U: \forall x. \text{sources-aux } (?cs' @ ?cs'') vs_1 s_1 f x \subseteq$
 $\text{sources-out } (?cs' @ ?cs'') vs_1 s_1 f x$
by (*blast intro: subsetD [OF sources-aux-sources-out]*)
moreover have $\forall x. \text{sources-aux } ?cs' vs_1 s_1 f x \subseteq$
 $\text{sources-aux } (?cs' @ ?cs'') vs_1 s_1 f x$

by (*blast intro: subsetD* [*OF sources-aux-append*])
ultimately have $V: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f } x)\}$
 using *S* by *blast*
have $W: \bigcup \{\text{tags-aux } (?cs' @ ?cs'') \text{ vs}_1 \text{ s}_1 \text{ f } x \mid x. x \in S\} \subseteq ?T$
 (is $?T' \subseteq -$)
 by (*blast intro: subsetD* [*OF tags-aux-tags-out*])
moreover have $\bigcup \{\text{tags-aux } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f } x \mid x. x \in S\} \subseteq ?T'$
 (is $?T'' \subseteq -$)
 by (*blast intro: subsetD* [*OF tags-aux-append*])
ultimately have $?T'' \subseteq ?T$
 by *simp*
with *T* **have** $f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T'')$
 by (*rule eq-streams-subset*)
hence $X: (c_1', t_1, f', \text{vs}_1', \text{ws}_1') \rightarrow^* (\text{SKIP}, t_1', f', \text{vs}_1'', \text{ws}_1'')$
 using *R* and *V* and $\langle ?P1 \rangle$ by *simp*
have $Y: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } (?cs' @ ?cs'') \text{ vs}_1 \text{ s}_1 \text{ f } x)\}$
 using *S* and *U* by *blast*
have $Z: f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T')$
 using *T* and *W* by (*rule eq-streams-subset*)
have $S \subseteq \{x. s_1' = t_1' (\subseteq \text{sources-aux } ?cs'' \text{ vs } s_1' \text{ f } x)\}$
 by (*rule sources-aux-rhs* [*OF Y Z L* $\langle ?P2 \rangle$])
moreover have $f = f' (\subseteq \text{vs}, \text{vs}_1'',$
 $\bigcup \{\text{tags-aux } ?cs'' \text{ vs } s_1' \text{ f } x \mid x. x \in S\})$
 by (*rule tags-aux-rhs* [*OF Y Z L X* $\langle ?P1 \rangle$])
ultimately have *AA*:
 $(c_2, t_1', f', \text{vs}_1'', \text{ws}_1'') \rightarrow^* (c_2', t_2, f', \text{vs}_2', \text{ws}_2')$
 using *R* and $\langle ?P1' \rangle$ by *simp*
have $\forall x. \text{sources-out } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f } x \subseteq$
 $\text{sources-out } (?cs' @ ?cs'') \text{ vs}_1 \text{ s}_1 \text{ f } x$
 by (*blast intro: subsetD* [*OF sources-out-append*])
hence $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f } x)\}$
 using *S* by *blast*
moreover have $\bigcup \{\text{tags-out } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f } x \mid x. x \in S\} \subseteq ?T$
 (is $?T' \subseteq -$)
 by (*blast intro: subsetD* [*OF tags-out-append*])
with *T* **have** $f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T')$
 by (*rule eq-streams-subset*)
ultimately have *AB*: $[p \leftarrow \text{drop } (\text{length } \text{ws}_1) \text{ ws}. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1'') \text{ ws}_1''. \text{fst } p \in S]$
 using *R* and $\langle ?P3 \rangle$ by *simp*
have $S \subseteq \{x. s_1' = t_1' (\subseteq \text{sources-out } ?cs'' \text{ vs } s_1' \text{ f } x)\}$
 by (*rule sources-out-rhs* [*OF S T L* $\langle ?P2 \rangle$])
moreover have $f = f' (\subseteq \text{vs}, \text{vs}_1'',$
 $\bigcup \{\text{tags-out } ?cs'' \text{ vs } s_1' \text{ f } x \mid x. x \in S\})$
 by (*rule tags-out-rhs* [*OF S T L X* $\langle ?P1 \rangle$])
ultimately have $[p \leftarrow \text{drop } (\text{length } \text{ws}) \text{ ws}_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1'') \text{ ws}_2'. \text{fst } p \in S]$
 using *R* and $\langle ?P3' \rangle$ by *simp*
hence $[p \leftarrow \text{drop } (\text{length } \text{ws}_1) \text{ ws}_2. \text{fst } p \in S] =$

```

    [p ← drop (length ws₁') ws₂'. fst p ∈ S]
  by (rule small-steps-outputs [OF L M X AA AB])
}
ultimately show
  ok-flow-aux-1 c' c'' 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
  using N by auto
qed
qed (simp add: no-upd-append N O Q)
qed
next
fix s' p cfs cfs'
assume I: (c₁, s, f, vs₀, ws₀) →*{cfs} (SKIP, s', p)
hence (c₁, s, f, vs₀, ws₀) ⇒ (s', p)
  by (auto dest: small-stepsl-steps simp: big-iff-small)
hence J: s' ∈ Univ B (⊆ state ∩ Y)
  using G and D by (rule ctying2-approx)
assume (c₂, s', p) →*{cfs'} (c', s₁, f, vs₁, ws₁)
moreover obtain vs and ws where p = (f, vs, ws)
  using I by (blast dest: small-stepsl-stream)
ultimately have K: (c₂, s', f, vs, ws) →*{cfs'} (c', s₁, f, vs₁, ws₁)
  by simp
show ?thesis
  using C [OF G - H J K F] by simp
qed
qed

lemma ctying2-correct-aux-or:
  assumes
    A: (U, v) ⊨ c₁ OR c₂ (⊆ A, X) = Some (C, Y) and
    B: ∧ C Y c' c'' s s₁ s₂ vs₀ vs₁ vs₂ ws₀ ws₁ ws₂ cfs₁ cfs₂.
      (U, v) ⊨ c₁ (⊆ A, X) = Some (C, Y) ⇒
      s ∈ Univ A (⊆ state ∩ X) ⇒
      (c₁, s, f, vs₀, ws₀) →*{cfs₁} (c', s₁, f, vs₁, ws₁) ⇒
      (c', s₁, f, vs₁, ws₁) →*{cfs₂} (c'', s₂, f, vs₂, ws₂) ⇒
      ok-flow-aux U c' c'' s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂) and
    C: ∧ C Y c' c'' s s₁ s₂ vs₀ vs₁ vs₂ ws₀ ws₁ ws₂ cfs₁ cfs₂.
      (U, v) ⊨ c₂ (⊆ A, X) = Some (C, Y) ⇒
      s ∈ Univ A (⊆ state ∩ X) ⇒
      (c₂, s, f, vs₀, ws₀) →*{cfs₁} (c', s₁, f, vs₁, ws₁) ⇒
      (c', s₁, f, vs₁, ws₁) →*{cfs₂} (c'', s₂, f, vs₂, ws₂) ⇒
      ok-flow-aux U c' c'' s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂) and
    D: s ∈ Univ A (⊆ state ∩ X) and
    E: (c₁ OR c₂, s, f, vs₀, ws₀) →*{cfs₁} (c', s₁, f, vs₁, ws₁) and
    F: (c', s₁, f, vs₁, ws₁) →*{cfs₂} (c'', s₂, f, vs₂, ws₂)
  shows ok-flow-aux U c' c'' s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂)
proof -

```

from A obtain C_1 and Y_1 and C_2 and Y_2 where
 $G: (U, v) \models c_1 (\subseteq A, X) = \text{Some } (C_1, Y_1)$ **and**
 $H: (U, v) \models c_2 (\subseteq A, X) = \text{Some } (C_2, Y_2)$
by (*auto split: option.split-asm*)
have
 $(c', s_1, f, vs_1, ws_1) = (c_1 \text{ OR } c_2, s, f, vs_0, ws_0) \vee$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_1\} (c', s_1, f, vs_1, ws_1) \vee$
 $(c_2, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_1\} (c', s_1, f, vs_1, ws_1)$
(is $?P \vee ?Q \vee ?R$ **)**
using E by (*blast dest: small-steps-l-or*)
thus $?thesis$
proof (*rule disjE, erule-tac [2] disjE*)
assume $?P$
hence $(c_1 \text{ OR } c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2)$
using F by *simp*
hence
 $(c'', s_2, f, vs_2, ws_2) = (c_1 \text{ OR } c_2, s, f, vs_0, ws_0) \wedge$
 $flow\ cfs_2 = [] \vee$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow\ cfs_2 = flow\ (tl\ cfs_2) \vee$
 $(c_2, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow\ cfs_2 = flow\ (tl\ cfs_2)$
(is $?P' \vee -$ **)**
by (*rule small-steps-l-or*)
thus $?thesis$
proof (*rule disjE, erule-tac [2] disjE, erule-tac [2-3] conjE*)
assume $?P'$
with $\langle ?P \rangle$ **show** $?thesis$
by *fastforce*
next
assume
 $I: (c_1, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2)$ **and**
 $J: flow\ cfs_2 = flow\ (tl\ cfs_2)$
(is $?cs = ?cs'$ **)**
have $K: (c_1, s, f, vs_0, ws_0) \rightarrow^*\{[]\} (c_1, s, f, vs_0, ws_0)$
by *simp*
hence $L: ok\text{-flow-aux } U\ c_1\ c''\ s\ s_2\ f\ vs_0\ vs_2\ ws_0\ ws_2\ ?cs'$
by (*rule B [OF G D - I]*)
show $?thesis$
proof (*rule conjI, clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $ok\text{-flow-aux-1 } c_1\ c''\ c_2'\ s\ t_1\ t_2\ f\ f'$
 $vs_0\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs' \wedge$
 $ok\text{-flow-aux-2 } s\ s_2\ t_1\ t_2\ f\ f'\ vs_0\ vs_1'\ ?cs' \wedge$
 $ok\text{-flow-aux-3 } s\ t_1\ f\ f'\ vs_0\ vs_1'\ ws_0\ ws_1'\ ws_2\ ws_2'\ ?cs'$
(is $?P1 \wedge ?P2 \wedge ?P3$ **)**
using L by *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*

show $\exists c_2' t_2 vs_2' ws_2'$.
ok-flow-aux-1 $c' c'' c_2' s_1 t_1 t_2 ff'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_2 ff' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 ff' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
proof (*rule exI* [*of* - c_2], *rule exI* [*of* - t_2],
rule exI [*of* - vs_2], *rule exI* [*of* - ws_2])
{
 fix S
 assume
 $S \neq \{\}$ **and**
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs' vs_1 s_1 f x)\}$ **and**
 $f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags-aux } ?cs' vs_1 s_1 f x \mid x. x \in S\})$
 hence
 $(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c'' = \text{SKIP}) = (c_2' = \text{SKIP}) \wedge$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p \in S] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p \in S]$
 (is $?Q1 \wedge ?Q2 \wedge ?Q3$ **)**
 using $\langle ?P \rangle$ **and** $\langle ?P1 \rangle$ **by** *simp*
 hence $?Q1$ **and** $?Q2$ **and** $?Q3$ **by** *auto*
 moreover have $(c_1 \text{ OR } c_2, t_1, f', vs_1', ws_1') \rightarrow$
 $(c_1, t_1, f', vs_1', ws_1') ..$
 hence $(c', t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$
 using $\langle ?P \rangle$ **and** $\langle ?Q1 \rangle$ **by** (*blast intro: star-trans*)
 ultimately have $?this \wedge ?Q2 \wedge ?Q3$
 by *simp*
}

moreover {
 fix S
 assume
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } ?cs' vs_1 s_1 f x)\}$ **and**
 $f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags } ?cs' vs_1 s_1 f x \mid x. x \in S\})$
 hence $s_2 = t_2 (\subseteq S)$
 using $\langle ?P \rangle$ **and** $\langle ?P2 \rangle$ **by** *blast*
}

moreover {
 fix S
 assume
 $S \neq \{\}$ **and**
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } ?cs' vs_1 s_1 f x)\}$ **and**
 $f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags-out } ?cs' vs_1 s_1 f x \mid x. x \in S\})$
 hence $[p \leftarrow \text{drop } (\text{length } ws_1) ws_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } ws_1') ws_2'. \text{fst } p \in S]$
 using $\langle ?P \rangle$ **and** $\langle ?P3 \rangle$ **by** *simp*
}

ultimately show
ok-flow-aux-1 $c' c'' c_2' s_1 t_1 t_2 ff'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$

$ok\text{-}flow\text{-}aux\text{-}2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
 $ok\text{-}flow\text{-}aux\text{-}3\ s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
using J **by** *auto*
qed
qed (*simp add: B [OF G D K I] J*)
next
assume
 $I: (c_2, s, f, vs_0, ws_0) \rightarrow*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2)$ **and**
 $J: flow\ cfs_2 = flow\ (tl\ cfs_2)$
 $(is\ ?cs = ?cs')$
have $K: (c_2, s, f, vs_0, ws_0) \rightarrow*\{\}\ (c_2, s, f, vs_0, ws_0)$
by *simp*
hence $L: ok\text{-}flow\text{-}aux\ U\ c_2\ c''\ s\ s_2\ f\ vs_0\ vs_2\ ws_0\ ws_2\ ?cs'$
by (*rule C [OF H D - I]*)
show *?thesis*
proof (*rule conjI, clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $ok\text{-}flow\text{-}aux\text{-}1\ c_2\ c''\ c_2'\ s\ t_1\ t_2\ f\ f'$
 $vs_0\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs' \wedge$
 $ok\text{-}flow\text{-}aux\text{-}2\ s\ s_2\ t_1\ t_2\ f\ f'\ vs_0\ vs_1'\ ?cs' \wedge$
 $ok\text{-}flow\text{-}aux\text{-}3\ s\ t_1\ f\ f'\ vs_0\ vs_1'\ ws_0\ ws_1'\ ws_2\ ws_2'\ ?cs'$
 $(is\ ?P1 \wedge ?P2 \wedge ?P3)$
using L **by** *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*
show $\exists\ c_2'\ t_2\ vs_2'\ ws_2'$.
 $ok\text{-}flow\text{-}aux\text{-}1\ c'\ c''\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
 $ok\text{-}flow\text{-}aux\text{-}2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
 $ok\text{-}flow\text{-}aux\text{-}3\ s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (*rule exI [of - c_2']*, *rule exI [of - t_2]*,
rule exI [of - vs_2'], *rule exI [of - ws_2']*)
{
fix S
assume
 $S \neq \{\}$ **and**
 $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-}aux\ ?cs'\ vs_1\ s_1\ f\ x)\}$ **and**
 $f = f' (\subseteq vs_1, vs_1', \bigcup \{tags\text{-}aux\ ?cs'\ vs_1\ s_1\ f\ x \mid x. x \in S\})$
hence
 $(c_2, t_1, f', vs_1', ws_1') \rightarrow* (c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c'' = SKIP) = (c_2' = SKIP) \wedge$
 $map\ fst\ [p \leftarrow drop\ (length\ vs_1)\ vs_2.\ fst\ p \in S] =$
 $map\ fst\ [p \leftarrow drop\ (length\ vs_1')\ vs_2'.\ fst\ p \in S]$
 $(is\ ?Q1 \wedge ?Q2 \wedge ?Q3)$
using $\langle ?P \rangle$ **and** $\langle ?P1 \rangle$ **by** *simp*
hence $?Q1$ **and** $?Q2$ **and** $?Q3$ **by** *auto*
moreover **have** $(c_1\ OR\ c_2, t_1, f', vs_1', ws_1') \rightarrow$
 $(c_2, t_1, f', vs_1', ws_1')$ **..**
hence $(c', t_1, f', vs_1', ws_1') \rightarrow* (c_2', t_2, f', vs_2', ws_2')$

```

    using ⟨?P⟩ and ⟨?Q1⟩ by (blast intro: star-trans)
    ultimately have ?this ∧ ?Q2 ∧ ?Q3
      by simp
  }
  moreover {
    fix S
    assume
      S ⊆ {x. s1 = t1 (⊆ sources ?cs' vs1 s1 f x)} and
      f = f' (⊆ vs1, vs1', ∪ {tags ?cs' vs1 s1 f x | x. x ∈ S})
    hence s2 = t2 (⊆ S)
      using ⟨?P⟩ and ⟨?P2⟩ by blast
  }
  moreover {
    fix S
    assume
      S ≠ {} and
      S ⊆ {x. s1 = t1 (⊆ sources-out ?cs' vs1 s1 f x)} and
      f = f' (⊆ vs1, vs1', ∪ {tags-out ?cs' vs1 s1 f x | x. x ∈ S})
    hence [p ← drop (length ws1) ws2. fst p ∈ S] =
      [p ← drop (length ws1') ws2'. fst p ∈ S]
      using ⟨?P⟩ and ⟨?P3⟩ by simp
  }
  ultimately show
    ok-flow-aux-1 c' c'' 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
      using J by auto
  qed
qed (simp add: C [OF H D K I] J)
qed
next
  assume ?Q
  thus ?thesis
    by (rule B [OF G D - F])
next
  assume ?R
  thus ?thesis
    by (rule C [OF H D - F])
qed
qed

```

lemma *ctyping2-correct-aux-if*:

assumes

A: $(U, v) \models \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ (} \subseteq A, X \text{) = Some (C, Y) and}$
 B: $\bigwedge U' p B_1 B_2 C_1 Y_1 c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$
 $(U', p) = (\text{insert (Univ? A X, bvars b) } U, \models b \text{ (} \subseteq A, X \text{)}) \implies$
 $(B_1, B_2) = p \implies$
 $(U', v) \models c_1 \text{ (} \subseteq B_1, X \text{) = Some (C}_1, Y_1) \implies$

$s \in Univ\ B_1 (\subseteq state \cap X) \implies$
 $(c_1, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$
ok-flow-aux $U' c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$ (*flow cfs₂*) **and**
 $C: \bigwedge U' p B_1 B_2 C_2 Y_2 c' c'' s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$
 $(U', p) = (insert (Univ? A X, bvars b) U, \models b (\subseteq A, X)) \implies$
 $(B_1, B_2) = p \implies$
 $(U', v) \models c_2 (\subseteq B_2, X) = Some (C_2, Y_2) \implies$
 $s \in Univ\ B_2 (\subseteq state \cap X) \implies$
 $(c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c', s_1, f, vs_1, ws_1) \implies$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c'', s_2, f, vs_2, ws_2) \implies$
ok-flow-aux $U' c' c'' s_1 s_2 f vs_1 vs_2 ws_1 ws_2$ (*flow cfs₂*) **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 cfs₂*)
proof –
let $?U' = insert (Univ? A X, bvars b) U$
from A **obtain** B_1 **and** B_2 **and** C_1 **and** C_2 **and** Y_1 **and** Y_2 **where**
 $G: \models b (\subseteq A, X) = (B_1, B_2)$ **and**
 $H: (?U', v) \models c_1 (\subseteq B_1, X) = Some (C_1, Y_1)$ **and**
 $I: (?U', v) \models c_2 (\subseteq B_2, X) = Some (C_2, Y_2)$
by (*auto split: option.split-asm prod.split-asm*)
have
 $(c', s_1, f, vs_1, ws_1) = (IF\ b\ THEN\ c_1\ ELSE\ c_2, s, f, vs_0, ws_0) \vee$
 $bval\ b\ s \wedge (c_1, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_1\} (c', s_1, f, vs_1, ws_1) \vee$
 $\neg\ bval\ b\ s \wedge (c_2, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_1\} (c', s_1, f, vs_1, ws_1)$
(is $?P \vee -$)
using E **by** (*blast dest: small-steps-l-if*)
thus $?thesis$
proof (*rule disjE, erule-tac [2] disjE, erule-tac [2-3] conjE*)
assume $?P$
hence $(IF\ b\ THEN\ c_1\ ELSE\ c_2, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\}$
 $(c'', s_2, f, vs_2, ws_2)$
using F **by** *simp*
hence
 $(c'', s_2, f, vs_2, ws_2) = (IF\ b\ THEN\ c_1\ ELSE\ c_2, s, f, vs_0, ws_0) \wedge$
 $flow\ cfs_2 = [] \vee$
 $bval\ b\ s \wedge (c_1, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow\ cfs_2 = \langle bvars\ b \rangle \# flow (tl\ cfs_2) \vee$
 $\neg\ bval\ b\ s \wedge (c_2, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow\ cfs_2 = \langle bvars\ b \rangle \# flow (tl\ cfs_2)$
(is $?P' \vee -$)
by (*rule small-steps-l-if*)
thus $?thesis$
proof (*rule disjE, erule-tac [2] disjE, (erule-tac [2-3] conjE)+*)
assume $?P'$
with $\langle ?P \rangle$ **show** $?thesis$

by *fastforce*
next
assume
J: *bval b s* **and**
K: $(c_1, s, f, vs_0, ws_0) \rightarrow^* \{tl\ cfs_2\} (c'', s_2, f, vs_2, ws_2)$ **and**
L: $flow\ cfs_2 = \langle bvars\ b \rangle \# flow\ (tl\ cfs_2)$
(is $?cs = - \# ?cs'$ **)**
have *M*: $s \in Univ\ B_1 (\subseteq state \cap X)$
using *J* **by** (*insert btyping2-approx* [*OF G D*], *simp*)
have *N*: $(c_1, s, f, vs_0, ws_0) \rightarrow^* \{\}\ (c_1, s, f, vs_0, ws_0)$
by *simp*
show *?thesis*
proof (*rule conjI, clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
ok-flow-aux-1 $c'\ c''\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
ok-flow-aux-2 $s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (*cases bval b t1*)
assume *O*: *bval b t1*
have *ok-flow-aux ?U'* $c_1\ c''\ s\ s_2\ f\ vs_0\ vs_2\ ws_0\ ws_2\ ?cs'$
using *B* [*OF - - H M N K*] **and** *G* **by** *simp*
then obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
ok-flow-aux-1 $c_1\ c''\ c_2'\ s\ t_1\ t_2\ f\ f'$
 $vs_0\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs' \wedge$
ok-flow-aux-2 $s\ s_2\ t_1\ t_2\ f\ f'\ vs_0\ vs_1'\ ?cs' \wedge$
ok-flow-aux-3 $s\ t_1\ f\ f'\ vs_0\ vs_1'\ ws_0\ ws_1'\ ws_2\ ws_2'\ ?cs'$
(is $?P1 \wedge ?P2 \wedge ?P3$ **)**
by *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*
show *?thesis*
proof (*rule exI* [*of - c2*'], *rule exI* [*of - t2*],
rule exI [*of - vs2*'], *rule exI* [*of - ws2*'])
{
fix *S*
assume
P: $S \neq \{\}$ **and**
Q: $S \subseteq \{x. s_1 = t_1$
 $(\subseteq sources\ aux\ (\langle bvars\ b \rangle \# ?cs')\ vs_1\ s_1\ f\ x)\}$ **and**
R: $f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags\ aux\ (\langle bvars\ b \rangle \# ?cs')\ vs_1\ s_1\ f\ x \mid x. x \in S\})$
have $\forall x. sources\ aux\ ?cs'\ vs_1\ s_1\ f\ x \subseteq$
 $sources\ aux\ (\langle bvars\ b \rangle \# ?cs')\ vs_1\ s_1\ f\ x$
by (*blast intro: subsetD* [*OF sources-aux-observe-tl*])
hence $S \subseteq \{x. s_1 = t_1 (\subseteq sources\ aux\ ?cs'\ vs_1\ s_1\ f\ x)\}$
using *Q* **by** *blast*
moreover have $f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags\ aux\ ?cs'\ vs_1\ s_1\ f\ x \mid x. x \in S\})$

```

    using R by (simp add: tags-aux-observe-tl)
  ultimately have
    (c1, t1, f', vs1', ws1') →* (c2', t2, f', vs2', ws2') ∧
    (c'' = SKIP) = (c2' = SKIP) ∧
    map fst [p ← drop (length vs1) vs2. fst p ∈ S] =
    map fst [p ← drop (length vs1') vs2'. fst p ∈ S]
    (is ?Q1 ∧ ?Q2 ∧ ?Q3)
    using P and ⟨?P⟩ and ⟨?P1⟩ by simp
  hence ?Q1 and ?Q2 and ?Q3 by auto
  moreover have (IF b THEN c1 ELSE c2, t1, f', vs1', ws1') →
    (c1, t1, f', vs1', ws1')
    using O ..
  hence (c', t1, f', vs1', ws1') →* (c2', t2, f', vs2', ws2')
    using ⟨?P⟩ and ⟨?Q1⟩ by (blast intro: star-trans)
  ultimately have ?this ∧ ?Q2 ∧ ?Q3
    by simp
}
moreover {
  fix S
  assume
    P: S ⊆ {x. s1 = t1}
      (⊆ sources ((bvars b) # ?cs') vs1 s1 f x) and
    Q: f = f' (⊆ vs1, vs1',
      ⋃ {tags ((bvars b) # ?cs') vs1 s1 f x | x. x ∈ S})
  have ∀x. sources ?cs' vs1 s1 f x ⊆
    sources ((bvars b) # ?cs') vs1 s1 f x
    by (blast intro: subsetD [OF sources-observe-tl])
  hence S ⊆ {x. s1 = t1} (⊆ sources ?cs' vs1 s1 f x)
    using P by blast
  moreover have f = f' (⊆ vs1, vs1',
    ⋃ {tags ?cs' vs1 s1 f x | x. x ∈ S})
    using Q by (simp add: tags-observe-tl)
  ultimately have s2 = t2 (⊆ S)
    using ⟨?P⟩ and ⟨?P2⟩ by blast
}
moreover {
  fix S
  assume
    P: S ≠ {} and
    Q: S ⊆ {x. s1 = t1}
      (⊆ sources-out ((bvars b) # ?cs') vs1 s1 f x) and
    R: f = f' (⊆ vs1, vs1',
      ⋃ {tags-out ((bvars b) # ?cs') vs1 s1 f x | x. x ∈ S})
  have ∀x. sources-out ?cs' vs1 s1 f x ⊆
    sources-out ((bvars b) # ?cs') vs1 s1 f x
    by (blast intro: subsetD [OF sources-out-observe-tl])
  hence S ⊆ {x. s1 = t1} (⊆ sources-out ?cs' vs1 s1 f x)
    using Q by blast
  moreover have f = f' (⊆ vs1, vs1',

```

```

     $\bigcup \{ \text{tags-out } ?cs' \text{ vs}_1 \text{ s}_1 \text{ f x} \mid x. x \in S \}$ 
    using R by (simp add: tags-out-observe-tl)
    ultimately have  $[p \leftarrow \text{drop } (\text{length } ws_1) \text{ ws}_2. \text{fst } p \in S] =$ 
     $[p \leftarrow \text{drop } (\text{length } ws_1') \text{ ws}_2'. \text{fst } p \in S]$ 
    using P and  $\langle ?P \rangle$  and  $\langle ?P\exists \rangle$  by simp
  }
  ultimately show
    ok-flow-aux-1 c' c'' 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-flow-aux-2 s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs  $\wedge$ 
    ok-flow-aux-3 s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs
    using L by auto
qed
next
assume O:  $\neg \text{bval } b \text{ t}_1$ 
show ?thesis
proof (cases  $\exists S \neq \{\}$ .  $S \subseteq \{x. s_1 = t_1$ 
  ( $\subseteq \text{sources-aux } ((\text{bvars } b) \# ?cs') \text{ vs}_1 \text{ s}_1 \text{ f x})\}$ )
  from O have (IF b THEN c1 ELSE c2, t1, f', vs1', ws1')  $\rightarrow$ 
    (c2, t1, f', vs1', ws1') ..
  moreover assume  $\exists S \neq \{\}$ .  $S \subseteq \{x. s_1 = t_1$ 
    ( $\subseteq \text{sources-aux } ((\text{bvars } b) \# ?cs') \text{ vs}_1 \text{ s}_1 \text{ f x})\}$ 
  then obtain S where
    P:  $S \neq \{\}$  and
    Q:  $S \subseteq \{x. s = t_1$ 
      ( $\subseteq \text{sources-aux } ((\text{bvars } b) \# ?cs') \text{ vs}_1 \text{ s f x})\}$ 
    using  $\langle ?P \rangle$  by blast
  have R: Univ? A X: bvars b  $\rightsquigarrow \mid S$ 
    using Q and D by (rule sources-aux-bval, insert J O, simp)
  have  $\exists p. (c_2, t_1, f', vs_1', ws_1') \Rightarrow p$ 
    using I by (rule ctyping2-term, insert P R, auto)
  then obtain t2 and f'' and vs2' and ws2' where
    S:  $(c_2, t_1, f', vs_1', ws_1') \rightarrow^* (\text{SKIP}, t_2, f'', vs_2', ws_2')$ 
    by (auto simp: big-iff-small)
  ultimately have
     $(c', t_1, f', vs_1', ws_1') \rightarrow^* (\text{SKIP}, t_2, f', vs_2', ws_2')$ 
    (is ?Q1)
    using  $\langle ?P \rangle$  by (blast dest: small-steps-stream
      intro: star-trans)
  have T:  $(c_2, t_1, f', vs_1', ws_1') \Rightarrow (t_2, f'', vs_2', ws_2')$ 
    using S by (simp add: big-iff-small)
  show ?thesis
  proof (cases c'' = SKIP)
    assume c'' = SKIP
      (is ?Q2)
    show ?thesis
  proof (rule exI [of - SKIP], rule exI [of - t2],
    rule exI [of - vs2'], rule exI [of - ws2'])
    {

```

```

fix S
assume  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-aux ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x) $\}$ 
hence  $U: Univ? A X: bvars\ b \rightsquigarrow | S$ 
  using D by (rule sources-aux-bval, insert J O, simp)
hence [ $p \leftarrow drop$  ( $length\ vs_1'$ )  $vs_2'$ . fst  $p \in S$ ] = []
  using I and T by (blast dest: ctyping2-confine)
moreover have no-upd  $?cs' S$ 
  using B [OF - - H M N K] and G and U by simp
hence [ $p \leftarrow in-flow\ ?cs' vs_1 f$ . fst  $p \in S$ ] = []
  by (rule no-upd-in-flow)
moreover have  $vs_2 = vs_0 @ in-flow\ ?cs' vs_0 f$ 
  using K by (rule small-steps1-in-flow)
ultimately have [ $p \leftarrow drop$  ( $length\ vs_1$ )  $vs_2$ . fst  $p \in S$ ] =
  [ $p \leftarrow drop$  ( $length\ vs_1'$ )  $vs_2'$ . fst  $p \in S$ ]
  using  $\langle ?P \rangle$  by simp
hence  $?Q1 \wedge ?Q2 \wedge ?this$ 
  using  $\langle ?Q1 \rangle$  and  $\langle ?Q2 \rangle$  by simp
}
moreover {
  fix S
  assume  $U: S \subseteq \{x. s = t_1$ 
    ( $\subseteq$  sources ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x) $\}$ 
  moreover have
     $\forall x. sources-aux$  ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x  $\subseteq$ 
      sources ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x
    by (blast intro: subsetD [OF sources-aux-sources])
  ultimately have  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq$  sources-aux ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x) $\}$ 
    by blast
  hence  $V: Univ? A X: bvars\ b \rightsquigarrow | S$ 
    using D by (rule sources-aux-bval, insert J O, simp)
  hence  $t_1 = t_2 (\subseteq S)$ 
    using I and T by (blast dest: ctyping2-confine)
  moreover have W: no-upd  $?cs' S$ 
    using B [OF - - H M N K] and G and V by simp
  hence run-flow  $?cs' vs_0 s f = s (\subseteq S)$ 
    by (rule no-upd-run-flow)
  moreover have  $s_2 = run-flow\ ?cs' vs_0 s f$ 
    using K by (rule small-steps1-run-flow)
  moreover have
     $\forall x \in S. x \in sources$  ( $\langle$ bvars b $\rangle \# ?cs'$ )  $vs_1$  s f x
    by (rule no-upd-sources, simp add: W)
  hence  $s = t_1 (\subseteq S)$ 
    using U by blast
  ultimately have  $s_2 = t_2 (\subseteq S)$ 
    by simp
}
moreover {

```

```

fix  $S$ 
assume  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-out ( $\langle bvars\ b \rangle \# ?cs'$ )  $vs_1\ s\ f\ x$ )}
moreover have
 $\forall x. sources\ aux\ (\langle bvars\ b \rangle \# ?cs')\ vs_1\ s\ f\ x \subseteq$ 
  sources-out ( $\langle bvars\ b \rangle \# ?cs'$ )  $vs_1\ s\ f\ x$ 
  by (blast intro: subsetD [OF sources-aux-sources-out])
ultimately have  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-aux ( $\langle bvars\ b \rangle \# ?cs'$ )  $vs_1\ s\ f\ x$ )}
  by blast
hence  $U: Univ? A\ X: bvars\ b \rightsquigarrow | S$ 
  using  $D$  by (rule sources-aux-bval, insert J O, simp)
hence [ $p \leftarrow drop\ (length\ ws_1')\ ws_2'.\ fst\ p \in S$ ] = []
  using  $I$  and  $T$  by (blast dest: ctyping2-confine)
moreover have no-upd  $?cs'\ S$ 
  using  $B$  [OF - - H M N K] and  $G$  and  $U$  by simp
hence [ $p \leftarrow out\ flow\ ?cs'\ vs_1\ s\ f.\ fst\ p \in S$ ] = []
  by (rule no-upd-out-flow)
moreover have  $ws_2 = ws_0 @ out\ flow\ ?cs'\ vs_0\ s\ f$ 
  using  $K$  by (rule small-steps1-out-flow)
ultimately have
  [ $p \leftarrow drop\ (length\ ws_1)\ ws_2.\ fst\ p \in S$ ] =
  [ $p \leftarrow drop\ (length\ ws_1')\ ws_2'.\ fst\ p \in S$ ]
  using  $\langle ?P \rangle$  by simp
}
ultimately show
ok-flow-aux-1  $c'\ c''\ SKIP\ s_1\ t_1\ t_2\ f\ f'$ 
   $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$ 
ok-flow-aux-2  $s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$ 
ok-flow-aux-3  $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$ 
using  $L$  and  $\langle ?P \rangle$  by auto
qed
next
assume  $c'' \neq SKIP$ 
  (is  $?Q2$ )
show  $?thesis$ 
proof (rule exI [of - c'], rule exI [of - t_1],
  rule exI [of - vs_1'], rule exI [of - ws_1'])
  {
    fix  $S$ 
    assume  $S \subseteq \{x. s = t_1$ 
      ( $\subseteq$  sources-aux ( $\langle bvars\ b \rangle \# ?cs'$ )  $vs_1\ s\ f\ x$ )}
    hence  $Univ? A\ X: bvars\ b \rightsquigarrow | S$ 
      using  $D$  by (rule sources-aux-bval, insert J O, simp)
    hence no-upd  $?cs'\ S$ 
      using  $B$  [OF - - H M N K] and  $G$  by simp
    hence [ $p \leftarrow in\ flow\ ?cs'\ vs_1\ f.\ fst\ p \in S$ ] = []
      by (rule no-upd-in-flow)
    moreover have  $vs_2 = vs_0 @ in\ flow\ ?cs'\ vs_0\ f$ 

```

```

    using  $K$  by (rule small-steps1-in-flow)
  ultimately have
    [ $p \leftarrow \text{drop } (\text{length } vs_1) \text{ } vs_2. \text{fst } p \in S$ ] = []
    using  $\langle ?P \rangle$  by simp
  hence  $?Q2 \wedge ?this$ 
    using  $\langle ?Q2 \rangle$  by simp
}
moreover {
  fix  $S$ 
  assume  $U: S \subseteq \{x. s = t_1$ 
    ( $\subseteq \text{sources } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x)\}$ 
  moreover have
     $\forall x. \text{sources-aux } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x \subseteq$ 
       $\text{sources } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x$ 
    by (blast intro: subsetD [OF sources-aux-sources])
  ultimately have  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq \text{sources-aux } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x)\}$ 
    by blast
  hence  $Univ? \ A \ X: bvars \ b \rightsquigarrow | S$ 
    using  $D$  by (rule sources-aux-bval, insert  $J \ O, \ simp$ )
  hence  $V: \text{no-upd } ?cs' \ S$ 
    using  $B$  [OF - -  $H \ M \ N \ K$ ] and  $G$  by simp
  hence  $\text{run-flow } ?cs' \ vs_0 \ s \ f = s \ (\subseteq S)$ 
    by (rule no-upd-run-flow)
  moreover have  $s_2 = \text{run-flow } ?cs' \ vs_0 \ s \ f$ 
    using  $K$  by (rule small-steps1-run-flow)
  moreover have
     $\forall x \in S. x \in \text{sources } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x$ 
    by (rule no-upd-sources, simp add:  $V$ )
  hence  $s = t_1 \ (\subseteq S)$ 
    using  $U$  by blast
  ultimately have  $s_2 = t_1 \ (\subseteq S)$ 
    by simp
}
moreover {
  fix  $S$ 
  assume  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq \text{sources-out } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x)\}$ 
  moreover have
     $\forall x. \text{sources-aux } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x \subseteq$ 
       $\text{sources-out } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x$ 
    by (blast intro: subsetD [OF sources-aux-sources-out])
  ultimately have  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq \text{sources-aux } (\langle bvars \ b \rangle \# ?cs') \ vs_1 \ s \ f \ x)\}$ 
    by blast
  hence  $Univ? \ A \ X: bvars \ b \rightsquigarrow | S$ 
    using  $D$  by (rule sources-aux-bval, insert  $J \ O, \ simp$ )
  hence  $\text{no-upd } ?cs' \ S$ 
    using  $B$  [OF - -  $H \ M \ N \ K$ ] and  $G$  by simp

```

hence $[p \leftarrow \text{out-flow } ?cs' \text{ vs}_1 \text{ s f. fst } p \in S] = []$
by (rule *no-upd-out-flow*)
moreover have $ws_2 = ws_0 @ \text{out-flow } ?cs' \text{ vs}_0 \text{ s f}$
using K **by** (rule *small-steps1-out-flow*)
ultimately have
 $[p \leftarrow \text{drop } (\text{length } ws_1) \text{ ws}_2. \text{fst } p \in S] = []$
using $\langle ?P \rangle$ **by** *simp*
}
ultimately show
 $\text{ok-flow-aux-1 } c' c'' c' s_1 t_1 t_1 f f'$
 $\text{vs}_1 \text{ vs}_1' \text{ vs}_2 \text{ vs}_1' \text{ ws}_1' \text{ ws}_1' ?cs \wedge$
 $\text{ok-flow-aux-2 } s_1 s_2 t_1 t_1 f f' \text{ vs}_1 \text{ vs}_1' ?cs \wedge$
 $\text{ok-flow-aux-3 } s_1 t_1 f f' \text{ vs}_1 \text{ vs}_1' \text{ ws}_1 \text{ ws}_1' \text{ ws}_2 \text{ ws}_1' ?cs$
using L **and** $\langle ?P \rangle$ **by** *auto*
qed
qed
next
assume $\nexists S. S \neq \{\} \wedge S \subseteq \{x. s_1 = t_1$
 $(\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') \text{ vs}_1 \text{ s}_1 \text{ f } x)\}$
hence $O: \forall c_2' t_2 \text{ vs}_2' \text{ ws}_2'.$
 $\text{ok-flow-aux-1 } c' c'' c_2' s_1 t_1 t_2 f f'$
 $\text{vs}_1 \text{ vs}_1' \text{ vs}_2 \text{ vs}_2' \text{ ws}_1' \text{ ws}_2' ?cs \wedge$
 $\text{ok-flow-aux-2 } s_1 s_2 t_1 t_2 f f' \text{ vs}_1 \text{ vs}_1' ?cs \wedge$
 $\text{ok-flow-aux-3 } s_1 t_1 f f' \text{ vs}_1 \text{ vs}_1' \text{ ws}_1 \text{ ws}_1' \text{ ws}_2 \text{ ws}_2' ?cs$
using L **by** (*auto intro!*: *ok-flow-aux-degen*)
show *?thesis*
by (rule *exI [of - SKIP]*, rule *exI [of - $\lambda x. 0$]*,
rule *exI [of - []]*, rule *exI [of - []]*,
simp add: O [*rule-format*, *of SKIP $\lambda x. 0$ [] []*])
qed
qed
qed (*simp add*: B [*OF - - H M N K*] $G L$)
next
assume
 $J: \neg \text{bval } b \text{ s and}$
 $K: (c_2, s, f, \text{vs}_0, \text{ws}_0) \rightarrow * \{ \text{tl } cfs_2 \} (c'', s_2, f, \text{vs}_2, \text{ws}_2)$ **and**
 $L: \text{flow } cfs_2 = \langle \text{bvars } b \rangle \# \text{flow } (\text{tl } cfs_2)$
 $(\text{is } ?cs = - \# ?cs')$
have $M: s \in \text{Univ } B_2 (\subseteq \text{state} \cap X)$
using J **by** (*insert btyping2-approx [OF G D]*, *simp*)
have $N: (c_2, s, f, \text{vs}_0, \text{ws}_0) \rightarrow * \{ [] \} (c_2, s, f, \text{vs}_0, \text{ws}_0)$
by *simp*
show *?thesis*
proof (rule *conjI*, *clarify*)
fix $t_1 f' \text{ vs}_1' \text{ ws}_1'$
show $\exists c_2' t_2 \text{ vs}_2' \text{ ws}_2'.$
 $\text{ok-flow-aux-1 } c' c'' c_2' s_1 t_1 t_2 f f'$
 $\text{vs}_1 \text{ vs}_1' \text{ vs}_2 \text{ vs}_2' \text{ ws}_1' \text{ ws}_2' ?cs \wedge$
 $\text{ok-flow-aux-2 } s_1 s_2 t_1 t_2 f f' \text{ vs}_1 \text{ vs}_1' ?cs \wedge$

$ok\text{-flow-aux-3 } s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
proof (*cases* $bval\ b\ t_1$, *cases* $\exists S \neq \{\}$.
 $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-aux } (\langle bvars\ b \rangle \# ?cs') vs_1 s_1 f x)\}$)
assume $O: \neg bval\ b\ t_1$
have $ok\text{-flow-aux } ?U' c_2 c'' s s_2 f vs_0 vs_2 ws_0 ws_2 ?cs'$
using $C [OF - - I M N K]$ **and** G **by** *simp*
then obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $ok\text{-flow-aux-1 } c_2 c'' c_2' s t_1 t_2 f f'$
 $vs_0 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs' \wedge$
 $ok\text{-flow-aux-2 } s s_2 t_1 t_2 f f' vs_0 vs_1' ?cs' \wedge$
 $ok\text{-flow-aux-3 } s t_1 f f' vs_0 vs_1' ws_0 ws_1' ws_2 ws_2' ?cs'$
(is $?P1 \wedge ?P2 \wedge ?P3$)
by *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*
show *thesis*
proof (*rule* $exI [of - c_2]$, *rule* $exI [of - t_2]$,
rule $exI [of - vs_2]$, *rule* $exI [of - ws_2]$)
 $\{$
fix S
assume
 $P: S \neq \{\}$ **and**
 $Q: S \subseteq \{x. s_1 = t_1$
 $(\subseteq sources\text{-aux } (\langle bvars\ b \rangle \# ?cs') vs_1 s_1 f x)\}$ **and**
 $R: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags\text{-aux } (\langle bvars\ b \rangle \# ?cs') vs_1 s_1 f x \mid x. x \in S\})$
have $\forall x. sources\text{-aux } ?cs' vs_1 s_1 f x \subseteq$
 $sources\text{-aux } (\langle bvars\ b \rangle \# ?cs') vs_1 s_1 f x$
by (*blast intro: subsetD [OF sources-aux-observe-tl]*)
hence $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-aux } ?cs' vs_1 s_1 f x)\}$
using Q **by** *blast*
moreover have $f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags\text{-aux } ?cs' vs_1 s_1 f x \mid x. x \in S\})$
using R **by** (*simp add: tags-aux-observe-tl*)
ultimately have
 $(c_2, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c'' = SKIP) = (c_2' = SKIP) \wedge$
 $map\ fst [p \leftarrow drop (length\ vs_1)\ vs_2. fst\ p \in S] =$
 $map\ fst [p \leftarrow drop (length\ vs_1')\ vs_2'. fst\ p \in S]$
(is $?Q1 \wedge ?Q2 \wedge ?Q3$)
using P **and** $\langle ?P \rangle$ **and** $\langle ?P1 \rangle$ **by** *simp*
hence $?Q1$ **and** $?Q2$ **and** $?Q3$ **by** *auto*
moreover have (*IF* b *THEN* c_1 *ELSE* $c_2, t_1, f', vs_1', ws_1'$) \rightarrow
 $(c_2, t_1, f', vs_1', ws_1')$
using $O ..$
hence $(c', t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$
using $\langle ?P \rangle$ **and** $\langle ?Q1 \rangle$ **by** (*blast intro: star-trans*)
ultimately have $?this \wedge ?Q2 \wedge ?Q3$
by *simp*
 $\}$

```

moreover {
  fix  $S$ 
  assume
     $P: S \subseteq \{x. s_1 = t_1$ 
       $(\subseteq \text{sources } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x)\}$  and
     $Q: f = f' (\subseteq vs_1, vs_1',$ 
       $\bigcup \{\text{tags } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x \mid x. x \in S\})$ 
  have  $\forall x. \text{sources } ?cs' vs_1 s_1 f x \subseteq$ 
     $\text{sources } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x$ 
    by (blast intro: subsetD [OF sources-observe-til])
  hence  $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } ?cs' vs_1 s_1 f x)\}$ 
    using  $P$  by blast
  moreover have  $f = f' (\subseteq vs_1, vs_1',$ 
     $\bigcup \{\text{tags } ?cs' vs_1 s_1 f x \mid x. x \in S\})$ 
    using  $Q$  by (simp add: tags-observe-til)
  ultimately have  $s_2 = t_2 (\subseteq S)$ 
    using  $\langle ?P \rangle$  and  $\langle ?P2 \rangle$  by blast
}
moreover {
  fix  $S$ 
  assume
     $P: S \neq \{\}$  and
     $Q: S \subseteq \{x. s_1 = t_1$ 
       $(\subseteq \text{sources-out } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x)\}$  and
     $R: f = f' (\subseteq vs_1, vs_1',$ 
       $\bigcup \{\text{tags-out } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x \mid x. x \in S\})$ 
  have  $\forall x. \text{sources-out } ?cs' vs_1 s_1 f x \subseteq$ 
     $\text{sources-out } (\langle \text{bvars } b \rangle \# ?cs^\wedge) vs_1 s_1 f x$ 
    by (blast intro: subsetD [OF sources-out-observe-til])
  hence  $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } ?cs' vs_1 s_1 f x)\}$ 
    using  $Q$  by blast
  moreover have  $f = f' (\subseteq vs_1, vs_1',$ 
     $\bigcup \{\text{tags-out } ?cs' vs_1 s_1 f x \mid x. x \in S\})$ 
    using  $R$  by (simp add: tags-out-observe-til)
  ultimately have  $[p \leftarrow \text{drop } (\text{length } ws_1) ws_2. \text{fst } p \in S] =$ 
     $[p \leftarrow \text{drop } (\text{length } ws_1') ws_2'. \text{fst } p \in S]$ 
    using  $P$  and  $\langle ?P \rangle$  and  $\langle ?P3 \rangle$  by simp
}
ultimately show
  ok-flow-aux-1  $c' c'' c_2' s_1 t_1 t_2 f f'$ 
     $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$ 
  ok-flow-aux-2  $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$ 
  ok-flow-aux-3  $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$ 
  using  $L$  by auto
qed
next
assume  $O: \text{bval } b t_1$ 
hence (IF  $b$  THEN  $c_1$  ELSE  $c_2, t_1, f', vs_1', ws_1'$ )  $\rightarrow$ 
   $(c_1, t_1, f', vs_1', ws_1') ..$ 

```

moreover assume $\exists S \neq \{\}$.
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s_1 f x)\}$
then obtain S **where**
 $P: S \neq \{\}$ **and**
 $Q: S \subseteq \{x. s = t_1 (\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x)\}$
using $\langle ?P \rangle$ **by** *blast*
have $R: \text{Univ? } A \ X: \text{bvars } b \rightsquigarrow | S$
using Q **and** D **by** (*rule sources-aux-bval, insert J O, simp*)
have $\exists p. (c_1, t_1, f', vs_1', ws_1') \Rightarrow p$
using H **by** (*rule ctying2-term, insert P R, auto*)
then obtain t_2 **and** f'' **and** vs_2' **and** ws_2' **where**
 $S: (c_1, t_1, f', vs_1', ws_1') \rightarrow^* (\text{SKIP}, t_2, f'', vs_2', ws_2')$
by (*auto simp: big-iff-small*)
ultimately have
 $(c', t_1, f', vs_1', ws_1') \rightarrow^* (\text{SKIP}, t_2, f', vs_2', ws_2')$
(is $?Q1$ **)**
using $\langle ?P \rangle$ **by** (*blast dest: small-steps-stream intro: star-trans*)
have $T: (c_1, t_1, f', vs_1', ws_1') \Rightarrow (t_2, f'', vs_2', ws_2')$
using S **by** (*simp add: big-iff-small*)
show $?thesis$
proof (*cases* $c'' = \text{SKIP}$)
assume $c'' = \text{SKIP}$
(is $?Q2$ **)**
show $?thesis$
proof (*rule exI [of - SKIP], rule exI [of - t_2],*
rule exI [of - vs_2'], rule exI [of - ws_2'])
 $\{$
fix S
assume $S \subseteq \{x. s = t_1$
 $(\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x)\}$
hence $U: \text{Univ? } A \ X: \text{bvars } b \rightsquigarrow | S$
using D **by** (*rule sources-aux-bval, insert J O, simp*)
hence $[p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p \in S] = []$
using H **and** T **by** (*blast dest: ctying2-confine*)
moreover have *no-upd* $?cs' S$
using C [*OF - - I M N K*] **and** G **and** U **by** *simp*
hence $[p \leftarrow \text{in-flow } ?cs' vs_1 f. \text{fst } p \in S] = []$
by (*rule no-upd-in-flow*)
moreover have $vs_2 = vs_0 @ \text{in-flow } ?cs' vs_0 f$
using K **by** (*rule small-steps1-in-flow*)
ultimately have $[p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } vs_1') vs_2'. \text{fst } p \in S]$
using $\langle ?P \rangle$ **by** *simp*
hence $?Q1 \wedge ?Q2 \wedge ?this$
using $\langle ?Q1 \rangle$ **and** $\langle ?Q2 \rangle$ **by** *simp*
 $\}$
moreover $\{$
fix S
assume $U: S \subseteq \{x. s = t_1$

```

    ( $\subseteq$  sources ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x)}
moreover have  $\forall x$ . sources-aux ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x  $\subseteq$ 
  sources ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x
by (blast intro: subsetD [OF sources-aux-sources])
ultimately have  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-aux ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x)}
by blast
hence  $V: Univ? A X: bvars b \rightsquigarrow | S$ 
using  $D$  by (rule sources-aux-bval, insert  $J O$ , simp)
hence  $t_1 = t_2 (\subseteq S)$ 
using  $H$  and  $T$  by (blast dest: ctyping2-confine)
moreover have  $W: no-upd ?cs' S$ 
using  $C$  [OF - - I M N K] and  $G$  and  $V$  by simp
hence run-flow ?cs' vs0 s f = s ( $\subseteq S$ )
by (rule no-upd-run-flow)
moreover have  $s_2 = run-flow ?cs' vs_0 s f$ 
using  $K$  by (rule small-steps1-run-flow)
moreover have  $\forall x \in S. x \in sources (\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x
by (rule no-upd-sources, simp add:  $W$ )
hence  $s = t_1 (\subseteq S)$ 
using  $U$  by blast
ultimately have  $s_2 = t_2 (\subseteq S)$ 
by simp
}
moreover {
  fix  $S$ 
assume  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-out ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x)}
moreover have  $\forall x$ . sources-aux ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x  $\subseteq$ 
  sources-out ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x
by (blast intro: subsetD [OF sources-aux-sources-out])
ultimately have  $S \subseteq \{x. s = t_1$ 
  ( $\subseteq$  sources-aux ( $\langle$ bvars  $b$  $\rangle$  # ?cs') vs1 s f x)}
by blast
hence  $U: Univ? A X: bvars b \rightsquigarrow | S$ 
using  $D$  by (rule sources-aux-bval, insert  $J O$ , simp)
hence  $[p \leftarrow drop (length ws_1') ws_2'. fst p \in S] = []$ 
using  $H$  and  $T$  by (blast dest: ctyping2-confine)
moreover have no-upd ?cs'  $S$ 
using  $C$  [OF - - I M N K] and  $G$  and  $U$  by simp
hence  $[p \leftarrow out-flow ?cs' vs_1 s f. fst p \in S] = []$ 
by (rule no-upd-out-flow)
moreover have  $ws_2 = ws_0 @ out-flow ?cs' vs_0 s f$ 
using  $K$  by (rule small-steps1-out-flow)
ultimately have
 $[p \leftarrow drop (length ws_1) ws_2. fst p \in S] =$ 
 $[p \leftarrow drop (length ws_1') ws_2'. fst p \in S]$ 
using  $\langle ?P \rangle$  by simp
}

```

ultimately show

ok-flow-aux-1 $c' c'' \text{ SKIP } s_1 t_1 t_2 f f'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
using L and $\langle ?P \rangle$ by *auto*

qed

next

assume $c'' \neq \text{SKIP}$
(is $?Q2$)
show *?thesis*

proof (rule *exI* [of - c'], rule *exI* [of - t_1],
rule *exI* [of - vs_1'], rule *exI* [of - ws_1'])
{
fix S
assume $S \subseteq \{x. s = t_1$
 $(\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x)\}$
hence *Univ?* $A X: \text{bvars } b \rightsquigarrow | S$
using D by (rule *sources-aux-bval*, insert $J O$, *simp*)
hence *no-upd* $?cs' S$
using C [$OF - - I M N K$] and G by *simp*
hence $[p \leftarrow \text{in-flow } ?cs' vs_1 f. \text{fst } p \in S] = []$
by (rule *no-upd-in-flow*)
moreover have $vs_2 = vs_0 @ \text{in-flow } ?cs' vs_0 f$
using K by (rule *small-stepst-in-flow*)
ultimately have $[p \leftarrow \text{drop } (\text{length } vs_1) vs_2. \text{fst } p \in S] = []$
using $\langle ?P \rangle$ by *simp*
hence $?Q2 \wedge ?this$
using $\langle ?Q2 \rangle$ by *simp*
}
moreover {
fix S
assume $U: S \subseteq \{x. s = t_1$
 $(\subseteq \text{sources } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x)\}$
moreover have $\forall x. \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x \subseteq$
 $\text{sources } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x$
by (*blast intro: subsetD* [$OF \text{ sources-aux-sources}$])
ultimately have $S \subseteq \{x. s = t_1$
 $(\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x)\}$
by *blast*
hence *Univ?* $A X: \text{bvars } b \rightsquigarrow | S$
using D by (rule *sources-aux-bval*, insert $J O$, *simp*)
hence $V: \text{no-upd } ?cs' S$
using C [$OF - - I M N K$] and G by *simp*
hence *run-flow* $?cs' vs_0 s f = s (\subseteq S)$
by (rule *no-upd-run-flow*)
moreover have $s_2 = \text{run-flow } ?cs' vs_0 s f$
using K by (rule *small-stepst-run-flow*)
moreover have $\forall x \in S. x \in \text{sources } (\langle \text{bvars } b \rangle \# ?cs') vs_1 s f x$

```

    by (rule no-upd-sources, simp add: V)
  hence  $s = t_1 (\subseteq S)$ 
    using  $U$  by blast
  ultimately have  $s_2 = t_1 (\subseteq S)$ 
    by simp
}
moreover {
  fix  $S$ 
  assume  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq$  sources-out ( $\langle$ bvars  $b$  $\rangle \# ?cs'$ )  $vs_1 s f x$ )}
  moreover have  $\forall x. \text{sources-aux } (\langle$ bvars  $b$  $\rangle \# ?cs'$ )  $vs_1 s f x \subseteq$ 
    sources-out ( $\langle$ bvars  $b$  $\rangle \# ?cs'$ )  $vs_1 s f x$ 
    by (blast intro: subsetD [OF sources-aux-sources-out])
  ultimately have  $S \subseteq \{x. s = t_1$ 
    ( $\subseteq$  sources-aux ( $\langle$ bvars  $b$  $\rangle \# ?cs'$ )  $vs_1 s f x$ )}
    by blast
  hence Univ?  $A X: \text{bvars } b \rightsquigarrow | S$ 
    using  $D$  by (rule sources-aux-bval, insert  $J O$ , simp)
  hence no-upd  $?cs' S$ 
    using  $C$  [OF - -  $I M N K$ ] and  $G$  by simp
  hence  $[p \leftarrow \text{out-flow } ?cs' vs_1 s f. \text{fst } p \in S] = []$ 
    by (rule no-upd-out-flow)
  moreover have  $ws_2 = ws_0 @ \text{out-flow } ?cs' vs_0 s f$ 
    using  $K$  by (rule small-steps1-out-flow)
  ultimately have  $[p \leftarrow \text{drop } (\text{length } ws_1) ws_2. \text{fst } p \in S] = []$ 
    using  $\langle ?P \rangle$  by simp
}
ultimately show
  ok-flow-aux-1  $c' c'' c' s_1 t_1 t_1 f f'$ 
     $vs_1 vs_1' vs_2 vs_1' ws_1' ws_1' ?cs \wedge$ 
  ok-flow-aux-2  $s_1 s_2 t_1 t_1 f f' vs_1 vs_1' ?cs \wedge$ 
  ok-flow-aux-3  $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_1' ?cs$ 
  using  $L$  and  $\langle ?P \rangle$  by auto
qed
qed
next
assume  $\nexists S. S \neq \{\}$   $\wedge$ 
   $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } (\langle$ bvars  $b$  $\rangle \# ?cs'$ )  $vs_1 s_1 f x)\}$ 
  hence  $O: \forall c_2' t_2 vs_2' ws_2'$ .
    ok-flow-aux-1  $c' c'' c_2' s_1 t_1 t_2 f f'$ 
       $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$ 
    ok-flow-aux-2  $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$ 
    ok-flow-aux-3  $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$ 
    using  $L$  by (auto intro!: ok-flow-aux-degen)
  show ?thesis
    by (rule exI [of - SKIP], rule exI [of -  $\lambda x. 0$ ],
      rule exI [of -  $[]$ ], rule exI [of -  $[]$ ],
      simp add:  $O$  [rule-format, of SKIP  $\lambda x. 0 [] []$ ])
qed

```

qed (*simp add: C [OF - - I M N K] G L*)
 qed
 next
 assume *bval b s*
 hence *J: s ∈ Univ B₁ (⊆ state ∩ X)*
 by (*insert btyping2-approx [OF G D], simp*)
 assume *K: (c₁, s, f, vs₀, ws₀) →*{tl cfs₁} (c', s₁, f, vs₁, ws₁)*
 show *?thesis*
 using *B [OF - - H J K F] and G by simp*
 next
 assume \neg *bval b s*
 hence *J: s ∈ Univ B₂ (⊆ state ∩ X)*
 by (*insert btyping2-approx [OF G D], simp*)
 assume *K: (c₂, s, f, vs₀, ws₀) →*{tl cfs₁} (c', s₁, f, vs₁, ws₁)*
 show *?thesis*
 using *C [OF - - I J K F] and G by simp*
 qed
 qed

lemma *ctyping2-correct-aux-while:*

assumes

A: (U, v) ⊨ WHILE b DO c (⊆ A, X) = Some (B, W) and

B: $\bigwedge B_1 B_2 C Y B_1' B_2' D Z c_1 c_2 s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

(B₁, B₂) = ⊨ b (⊆ A, X) ⇒

(C, Y) = ⊢ c (⊆ B₁, X) ⇒

(B₁', B₂') = ⊨ b (⊆ C, Y) ⇒

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

B: W \rightsquigarrow UNIV ⇒

({ }, False) ⊨ c (⊆ B₁, X) = Some (D, Z) ⇒

s ∈ Univ B₁ (⊆ state ∩ X) ⇒

(c, s, f, vs₀, ws₀) →{cfs₁} (c₁, s₁, f, vs₁, ws₁) ⇒*

(c₁, s₁, f, vs₁, ws₁) →{cfs₂} (c₂, s₂, f, vs₂, ws₂) ⇒*

ok-flow-aux { } c₁ c₂ s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂) and

C: $\bigwedge B_1 B_2 C Y B_1' B_2' D' Z' c_1 c_2 s s_1 s_2 vs_0 vs_1 vs_2 ws_0 ws_1 ws_2 cfs_1 cfs_2.$

(B₁, B₂) = ⊨ b (⊆ A, X) ⇒

(C, Y) = ⊢ c (⊆ B₁, X) ⇒

(B₁', B₂') = ⊨ b (⊆ C, Y) ⇒

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

B: W \rightsquigarrow UNIV ⇒

({ }, False) ⊨ c (⊆ B₁', Y) = Some (D', Z') ⇒

s ∈ Univ B₁' (⊆ state ∩ Y) ⇒

(c, s, f, vs₀, ws₀) →{cfs₁} (c₁, s₁, f, vs₁, ws₁) ⇒*

(c₁, s₁, f, vs₁, ws₁) →{cfs₂} (c₂, s₂, f, vs₂, ws₂) ⇒*

ok-flow-aux { } c₁ c₂ s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂) and

D: s ∈ Univ A (⊆ state ∩ X) and

E: (WHILE b DO c, s, f, vs₀, ws₀) →{cfs₁} (c₁, s₁, f, vs₁, ws₁) and*

F: (c₁, s₁, f, vs₁, ws₁) →{cfs₂} (c₂, s₂, f, vs₂, ws₂)*

shows *ok-flow-aux U c₁ c₂ s₁ s₂ f vs₁ vs₂ ws₁ ws₂ (flow cfs₂)*

proof –

from A **obtain** $B_1 B_2 C Y B_1' B_2' D Z D' Z'$ **where**

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

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

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

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

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

$L: \forall (B, W) \in insert (Univ? A X \cup Univ? C Y, bvars b) U. B: W \rightsquigarrow UNIV$

by (*fastforce split: if-split-asm option.split-asm prod.split-asm*)

from $UnI1$ [$OF D$, of $Univ C (\subseteq state \cap Y)$] **and** E **and** F **show** *?thesis*

proof (*induction cfs₁ @ cfs₂ arbitrary: cfs₁ cfs₂ s vs₀ ws₀ c₁ s₁ vs₁ ws₁*

rule: length-induct)

fix $cfs_1 cfs_2 s vs_0 ws_0 c_1 s_1 vs_1 ws_1$

assume

$M: \forall cfs. length\ cfs < length\ (cfs_1\ @\ cfs_2) \longrightarrow$

$(\forall cfs' cfs''. cfs = cfs' @ cfs'' \longrightarrow$

$(\forall s. s \in Univ\ A (\subseteq state \cap X) \cup Univ\ C (\subseteq state \cap Y) \longrightarrow$

$(\forall vs_0\ ws_0\ c_1\ s_1\ vs_1\ ws_1.$

$(WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \rightarrow^*\{cfs'\} (c_1, s_1, f, vs_1, ws_1) \longrightarrow$

$(c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs''\} (c_2, s_2, f, vs_2, ws_2) \longrightarrow$

$ok-flow-aux\ U\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs''))$) **and**

$N: s \in Univ\ A (\subseteq state \cap X) \cup Univ\ C (\subseteq state \cap Y)$ **and**

$O: (c_1, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

assume $(WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$

hence

$(c_1, s_1, f, vs_1, ws_1) = (WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \wedge$

$flow\ cfs_1 = [] \vee$

$bval\ b\ s \wedge$

$(c;;\ WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_1\} (c_1, s_1, f, vs_1, ws_1) \wedge$

$flow\ cfs_1 = \langle bvars\ b \rangle \# flow\ (tl\ cfs_1) \vee$

$\neg\ bval\ b\ s \wedge$

$(c_1, s_1, f, vs_1, ws_1) = (SKIP, s, f, vs_0, ws_0) \wedge$

$flow\ cfs_1 = [\langle bvars\ b \rangle]$

by (*rule small-steps-l-while*)

thus $ok-flow-aux\ U\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ (flow\ cfs_2)$

proof (*rule disjE, erule-tac [2] disjE, erule-tac conjE,*

(erule-tac [2-3] conjE)+)

assume $P: (c_1, s_1, f, vs_1, ws_1) = (WHILE\ b\ DO\ c, s, f, vs_0, ws_0)$

hence $(WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \rightarrow^*\{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$

using O **by** *simp*

hence

$(c_2, s_2, f, vs_2, ws_2) = (WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \wedge$

$flow\ cfs_2 = [] \vee$

$bval\ b\ s \wedge$

$(c;;\ WHILE\ b\ DO\ c, s, f, vs_0, ws_0) \rightarrow^*\{tl\ cfs_2\}$

$(c_2, s_2, f, vs_2, ws_2) \wedge$

$flow\ cfs_2 = \langle bvars\ b \rangle \# flow\ (tl\ cfs_2) \vee$

$\neg\ bval\ b\ s \wedge$

$(c_2, s_2, f, vs_2, ws_2) = (SKIP, s, f, vs_0, ws_0) \wedge$

$flow\ cfs_2 = [\langle bvars\ b \rangle]$

by (*rule small-stepsl-while*)
thus *?thesis*
proof (*rule disjE*, *erule-tac* [2] *disjE*, *erule-tac conjE*,
(*erule-tac* [2-3] *conjE*)+)
assume
 $(c_2, s_2, f, vs_2, ws_2) = (\text{WHILE } b \text{ DO } c, s, f, vs_0, ws_0)$ **and**
 $\text{flow } cfs_2 = []$
thus *?thesis*
using P **by** *fastforce*
next
assume
 $Q: \text{bval } b \text{ } s$ **and**
 $R: \text{flow } cfs_2 = \langle \text{bvars } b \rangle \# \text{flow } (tl \ cfs_2)$
 $(\text{is } ?cs = - \# ?cs')$
assume $(c;; \text{WHILE } b \text{ DO } c, s, f, vs_0, ws_0) \rightarrow^*\{tl \ cfs_2\}$
 $(c_2, s_2, f, vs_2, ws_2)$
hence
 $(\exists c' \ cfs.$
 $c_2 = c';; \text{WHILE } b \text{ DO } c \wedge$
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c', s_2, f, vs_2, ws_2) \wedge$
 $?cs' = \text{flow } cfs) \vee$
 $(\exists p \ cfs' \ cfs''.$
 $\text{length } cfs'' < \text{length } (tl \ cfs_2) \wedge$
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs'\} (\text{SKIP}, p) \wedge$
 $(\text{WHILE } b \text{ DO } c, p) \rightarrow^*\{cfs''\} (c_2, s_2, f, vs_2, ws_2) \wedge$
 $?cs' = \text{flow } cfs' @ \text{flow } cfs'')$
by (*rule small-stepsl-seq*)
thus *?thesis*
apply (*rule disjE*)
apply (*erule exE*)+
apply (*erule conjE*)+
subgoal for $c' \ cfs$
proof –
assume
 $S: c_2 = c';; \text{WHILE } b \text{ DO } c$ **and**
 $T: (c, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c', s_2, f, vs_2, ws_2)$ **and**
 $U: ?cs' = \text{flow } cfs$
have $V: (c, s, f, vs_0, ws_0) \rightarrow^*\{[]\} (c, s, f, vs_0, ws_0)$
by *simp*
from N **have**
 $ok\text{-flow-aux } \{ \} \ c \ c' \ s \ s_2 \ f \ vs_0 \ vs_2 \ ws_0 \ ws_2 \ (\text{flow } cfs)$
proof
assume $W: s \in \text{Univ } A \ (\subseteq \text{state} \cap X)$
have $X: s \in \text{Univ } B_1 \ (\subseteq \text{state} \cap X)$
using Q **by** (*insert btyping2-approx* [*OF G W*], *simp*)
show *?thesis*
by (*rule B* [*OF G* [*symmetric*] *H* [*symmetric*] *I* [*symmetric*]
 $L \ J \ X \ V \ T$])
next

assume $W: s \in \text{Univ } C (\subseteq \text{state} \cap Y)$
have $X: s \in \text{Univ } B_1' (\subseteq \text{state} \cap Y)$
using Q **by** (*insert btyping2-approx* [$OF\ I\ W$], *simp*)
show $?thesis$
by (*rule* C [$OF\ G$ [*symmetric*] H [*symmetric*] I [*symmetric*]
 $L\ K\ X\ V\ T$])

qed
hence $W: \text{ok-flow-aux } \{\} c\ c'\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ ?cs'$
using P **and** U **by** *simp*
show $?thesis$
proof (*rule* *conjI*, *clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $\text{ok-flow-aux-1 } c\ c'\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs' \wedge$
 $\text{ok-flow-aux-2 } s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs' \wedge$
 $\text{ok-flow-aux-3 } s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs'$
(is $?P1 \wedge ?P2 \wedge ?P3$ **)**
using W **by** *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
 $\text{ok-flow-aux-1 } c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
 $\text{ok-flow-aux-2 } s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
 $\text{ok-flow-aux-3 } s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (*rule* *exI* [*of* - c_2' ;; *WHILE* $b\ DO\ c$], *rule* *exI* [*of* - t_2],
rule *exI* [*of* - vs_2'], *rule* *exI* [*of* - ws_2'])
{
fix S
assume
 $X: S \neq \{\}$ **and**
 $Y: S \subseteq \{x. s_1 = t_1$
 $(\subseteq \text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs')\ vs_1\ s_1\ f\ x)\}$ **and**
 $Z: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{\text{tags-aux } (\langle \text{bvars } b \rangle \# ?cs')\ vs_1\ s_1\ f\ x \mid x. x \in S\})$
have $\forall x. \text{sources-aux } ?cs'\ vs_1\ s_1\ f\ x \subseteq$
 $\text{sources-aux } (\langle \text{bvars } b \rangle \# ?cs')\ vs_1\ s_1\ f\ x$
by (*blast intro: subsetD* [$OF\ \text{sources-aux-observe-tl}$])
hence $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs'\ vs_1\ s_1\ f\ x)\}$
using Y **by** *blast*
moreover **have** $f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{\text{tags-aux } ?cs'\ vs_1\ s_1\ f\ x \mid x. x \in S\})$
using Z **by** (*simp add: tags-aux-observe-tl*)
ultimately **have**
 $(c, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1)\ vs_2. \text{fst } p \in S] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1')\ vs_2'. \text{fst } p \in S]$
(is $?Q1 \wedge ?Q2$ **)**
using X **and** $\langle ?P1 \rangle$ **by** *simp*

```

hence  $?Q1$  and  $?Q2$  by auto
have  $s_1 = t_1$  ( $\subseteq$  bvars  $b$ )
  by (rule eq-states-while [OF Y X], insert L N P, simp+)
hence bval  $b$   $t_1$ 
  using  $P$  and  $Q$  by (blast dest: bvars-bval)
hence (WHILE  $b$  DO  $c$ ,  $t_1$ ,  $f'$ ,  $vs_1'$ ,  $ws_1'$ )  $\rightarrow$ 
  ( $c$ ; WHILE  $b$  DO  $c$ ,  $t_1$ ,  $f'$ ,  $vs_1'$ ,  $ws_1'$ )  $\dots$ 
hence ( $c_1$ ,  $t_1$ ,  $f'$ ,  $vs_1'$ ,  $ws_1'$ )  $\rightarrow^*$ 
  ( $c_2'$ ; WHILE  $b$  DO  $c$ ,  $t_2$ ,  $f'$ ,  $vs_2'$ ,  $ws_2'$ )
  using  $P$  and  $\langle ?Q1 \rangle$  by (blast intro: star-seq2 star-trans)
hence  $?this \wedge ?Q2$ 
  using  $\langle ?Q2 \rangle$  by simp
}
moreover {
  fix  $S$ 
  assume
     $X: S \subseteq \{x. s_1 = t_1$ 
      ( $\subseteq$  sources ( $\langle \text{bvars } b \rangle \# ?cs'$ )  $vs_1$   $s_1$   $f$   $x\})$  and
       $Y: f = f'$  ( $\subseteq$   $vs_1, vs_1'$ ,
         $\bigcup \{tags \langle \text{bvars } b \rangle \# ?cs' vs_1 s_1 f x \mid x. x \in S\})$ )
    have  $\forall x. sources ?cs' vs_1 s_1 f x \subseteq$ 
      sources ( $\langle \text{bvars } b \rangle \# ?cs'$ )  $vs_1$   $s_1$   $f$   $x$ 
      by (blast intro: subsetD [OF sources-observe-tl])
    hence  $S \subseteq \{x. s_1 = t_1$  ( $\subseteq$  sources  $?cs' vs_1 s_1 f x\})$ 
      using  $X$  by blast
    moreover have  $f = f'$  ( $\subseteq$   $vs_1, vs_1'$ ,
       $\bigcup \{tags ?cs' vs_1 s_1 f x \mid x. x \in S\})$ 
      using  $Y$  by (simp add: tags-observe-tl)
    ultimately have  $s_2 = t_2$  ( $\subseteq$   $S$ )
      using  $\langle ?P2 \rangle$  by blast
}
moreover {
  fix  $S$ 
  assume
     $X: S \neq \{\}$  and
     $Y: S \subseteq \{x. s_1 = t_1$ 
      ( $\subseteq$  sources-out ( $\langle \text{bvars } b \rangle \# ?cs'$ )  $vs_1$   $s_1$   $f$   $x\})$  and
       $Z: f = f'$  ( $\subseteq$   $vs_1, vs_1'$ ,
         $\bigcup \{tags-out \langle \text{bvars } b \rangle \# ?cs' vs_1 s_1 f x \mid x. x \in S\})$ )
    have  $\forall x. sources-out ?cs' vs_1 s_1 f x \subseteq$ 
      sources-out ( $\langle \text{bvars } b \rangle \# ?cs'$ )  $vs_1$   $s_1$   $f$   $x$ 
      by (blast intro: subsetD [OF sources-out-observe-tl])
    hence  $S \subseteq \{x. s_1 = t_1$  ( $\subseteq$  sources-out  $?cs' vs_1 s_1 f x\})$ 
      using  $Y$  by blast
    moreover have  $f = f'$  ( $\subseteq$   $vs_1, vs_1'$ ,
       $\bigcup \{tags-out ?cs' vs_1 s_1 f x \mid x. x \in S\})$ 
      using  $Z$  by (simp add: tags-out-observe-tl)
    ultimately have  $[p \leftarrow drop (length ws_1) ws_2. fst p \in S] =$ 
       $[p \leftarrow drop (length ws_1') ws_2'. fst p \in S]$ 
}

```

```

    using  $X$  and  $\langle ?P3 \rangle$  by simp
  }
  ultimately show
    ok-flow-aux-1  $c_1 c_2 (c_2'; \text{WHILE } b \text{ DO } c) s_1 t_1 t_2 f f'$ 
       $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$ 
    ok-flow-aux-2  $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$ 
    ok-flow-aux-3  $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$ 
    using  $R$  and  $S$  by auto
  qed
  qed (insert L, auto simp: no-upd-empty)
  qed
  apply (erule exE)
  apply (erule conjE)
  subgoal for  $p cfs' cfs''$ 
  proof -
    assume  $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs'\} (\text{SKIP}, p)$ 
    moreover from this obtain  $s_1'$  and  $vs$  and  $ws$  where
       $S: p = (s_1', f, vs, ws)$ 
    by (blast dest: small-steps1-stream)
    ultimately have
       $T: (c, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (\text{SKIP}, s_1', f, vs, ws)$ 
    using  $P$  by simp
    assume  $(\text{WHILE } b \text{ DO } c, p) \rightarrow^*\{cfs''\} (c_2, s_2, f, vs_2, ws_2)$ 
    with  $S$  have
       $U: (\text{WHILE } b \text{ DO } c, s_1', f, vs, ws) \rightarrow^*\{cfs''\}$ 
       $(c_2, s_2, f, vs_2, ws_2)$ 
    by simp
    assume  $V: ?cs' = \text{flow } cfs' @ \text{flow } cfs''$ 
    (is  $(- :: \text{flow}) = ?cs_1' @ ?cs_2'$ )
    have  $W: (c, s_1, f, vs_1, ws_1) \rightarrow^*\{\}\} (c, s_1, f, vs_1, ws_1)$ 
    by simp
    from  $N$  have ok-flow-aux  $\{ \} c \text{SKIP } s_1 s_1' f vs_1 vs ws_1 ws ?cs_1'$ 
  proof
    assume  $X: s \in \text{Univ } A (\subseteq \text{state} \cap X)$ 
    have  $Y: s_1 \in \text{Univ } B_1 (\subseteq \text{state} \cap X)$ 
    using  $P$  and  $Q$  by (insert btyping2-approx [OF G X], simp)
    show ?thesis
    by (rule B [OF G [symmetric] H [symmetric] I [symmetric]
       $L J Y W T$ )
  next
    assume  $X: s \in \text{Univ } C (\subseteq \text{state} \cap Y)$ 
    have  $Y: s_1 \in \text{Univ } B_1' (\subseteq \text{state} \cap Y)$ 
    using  $P$  and  $Q$  by (insert btyping2-approx [OF I X], simp)
    show ?thesis
    by (rule C [OF G [symmetric] H [symmetric] I [symmetric]
       $L K Y W T$ )
  qed
  hence  $X: \text{ok-flow-aux } \{ \} c \text{SKIP } s_1 s_1' f vs_1 vs ws_1 ws ?cs_1'$ 
  using  $P$  by simp

```

assume $\text{length } cfs'' < \text{length } (tl \ cfs_2)$
hence $\text{length } (\square @ cfs'') < \text{length } (cfs_1 @ cfs_2)$
by *simp*
moreover have $\square @ cfs'' = \square @ cfs'' ..$
moreover from T **have** $(c, s, f, vs_0, ws_0) \Rightarrow (s_1', f, vs, ws)$
using P **by** (*auto dest: small-steps1-steps simp: big-iff-small*)
hence $s_1' \in \text{Univ } A (\subseteq \text{state} \cap X) \cup \text{Univ } C (\subseteq \text{state} \cap Y)$
by (*rule univ-states-while [OF - G H I J K Q N]*)
moreover have $(\text{WHILE } b \text{ DO } c, s_1', f, vs, ws) \rightarrow^* \{\square\}$
 $(\text{WHILE } b \text{ DO } c, s_1', f, vs, ws)$
by *simp*
ultimately have
 $Y: \text{ok-flow-aux } U (\text{WHILE } b \text{ DO } c) \ c_2 \ s_1' \ s_2 \ f \ vs \ vs_2 \ ws \ ws_2 \ ?cs_2'$
using U **by** (*rule M [rule-format]*)
show *?thesis*
proof (*rule conjI, clarify*)
fix $t_1 \ f' \ vs_1' \ ws_1'$
obtain c_1'' **and** t_1' **and** vs_1'' **and** ws_1'' **where**
 $\text{ok-flow-aux-1 } c \ \text{SKIP } c_1'' \ s_1 \ t_1 \ t_1' \ f \ f'$
 $vs_1 \ vs_1' \ vs \ vs_1'' \ ws_1' \ ws_1'' \ ?cs_1' \ \wedge$
 $\text{ok-flow-aux-2 } s_1 \ s_1' \ t_1 \ t_1' \ f \ f' \ vs_1 \ vs_1' \ ?cs_1' \ \wedge$
 $\text{ok-flow-aux-3 } s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws \ ws_1'' \ ?cs_1'$
 $(\text{is } - \ \wedge \ ?P2 \ \wedge \ ?P3)$
using X **by** *fastforce*
hence
 $\text{ok-flow-aux-1 } c \ \text{SKIP } \text{SKIP } s_1 \ t_1 \ t_1' \ f \ f'$
 $vs_1 \ vs_1' \ vs \ vs_1'' \ ws_1' \ ws_1'' \ ?cs_1'$
 $(\text{is } ?P1) \ \text{and } ?P2 \ \text{and } ?P3$ **by** *auto*
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $\text{ok-flow-aux-1 } (\text{WHILE } b \text{ DO } c) \ c_2 \ c_2' \ s_1' \ t_1' \ t_2 \ f \ f'$
 $vs \ vs_1'' \ vs_2 \ vs_2' \ ws_1'' \ ws_2' \ ?cs_2' \ \wedge$
 $\text{ok-flow-aux-2 } s_1' \ s_2 \ t_1' \ t_2 \ f \ f' \ vs \ vs_1'' \ ?cs_2' \ \wedge$
 $\text{ok-flow-aux-3 } s_1' \ t_1' \ f \ f' \ vs \ vs_1'' \ ws \ ws_1'' \ ws_2 \ ws_2' \ ?cs_2'$
 $(\text{is } ?P1' \ \wedge \ ?P2' \ \wedge \ ?P3')$
using Y **by** *fastforce*
hence $?P1'$ **and** $?P2'$ **and** $?P3'$ **by** *auto*
show $\exists c_2' \ t_2 \ vs_2' \ ws_2'$.
 $\text{ok-flow-aux-1 } c_1 \ c_2 \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f'$
 $vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \ \wedge$
 $\text{ok-flow-aux-2 } s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \ \wedge$
 $\text{ok-flow-aux-3 } s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$
proof (*rule exI [of - c_2'], rule exI [of - t_2],*
rule exI [of - vs_2'], rule exI [of - ws_2'])
 $\{$
fix S
assume
 $Z: S \neq \{\}$ **and**
 $AA: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux}$
 $(\text{bvars } b) \ \# \ ?cs_1' @ \ ?cs_2') \ vs_1 \ s_1 \ f \ x\}$ **and**

AB: $f = f' (\subseteq vs_1, vs_1', \bigcup \{tags\text{-aux} (\langle bvars \ b \rangle \# \ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x \mid x. x \in S\})$
have $\forall x. sources\text{-aux} (\ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x \subseteq sources\text{-aux} (\langle bvars \ b \rangle \# \ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x$
by (*blast intro: subsetD [OF sources-*aux-observe-tl*]*)
hence AC: $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-aux} (\ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x)\}$
using AA by blast
moreover have $\forall x. sources\text{-aux} \ ?cs_1' \ vs_1 \ s_1 \ f \ x \subseteq sources\text{-aux} (\ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x$
by (*blast intro: subsetD [OF sources-*aux-append*]*)
ultimately have
AD: $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-aux} \ ?cs_1' \ vs_1 \ s_1 \ f \ x)\}$
by blast
have AE: $f = f' (\subseteq vs_1, vs_1', \bigcup \{tags\text{-aux} (\ ?cs_1' \ @ \ ?cs_2') vs_1 \ s_1 \ f \ x \mid x. x \in S\})$
(is - = - (\subseteq -, -, ?T))
using AB by (simp add: tags-*aux-observe-tl*)
moreover have
 $\bigcup \{tags\text{-aux} \ ?cs_1' \ vs_1 \ s_1 \ f \ x \mid x. x \in S\} \subseteq \ ?T$
(is ?T' \subseteq -)
by (blast intro: subsetD [OF tags-*aux-append*])
ultimately have $f = f' (\subseteq vs_1, vs_1', \ ?T)$
by (rule eq-streams-subset)
hence
 $(c, t_1, f', vs_1', ws_1') \rightarrow^* (SKIP, t_1', f', vs_1'', ws_1'') \wedge$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1) \ vs. \ fst \ p \in S] =$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1') \ vs_1''. \ fst \ p \in S]$
(is ?Q1 \wedge ?Q2)
using Z and AD and $\langle \ ?P1 \ \rangle$ by simp
hence ?Q1 and ?Q2 by auto
have $S \subseteq \{x. s_1' = t_1' (\subseteq sources\text{-aux} \ ?cs_2' \ vs \ s_1' \ f \ x)\}$
by (rule sources-*aux-rhs* [OF AC AE T $\langle \ ?P2 \ \rangle$])
moreover have $f = f' (\subseteq vs, vs_1'', \bigcup \{tags\text{-aux} \ ?cs_2' \ vs \ s_1' \ f \ x \mid x. x \in S\})$
by (rule tags-*aux-rhs* [OF AC AE T $\langle \ ?Q1 \ \rangle \ \langle \ ?P1 \ \rangle$])
ultimately have
 $(WHILE \ b \ DO \ c, t_1', f', vs_1'', ws_1'') \rightarrow^*$
 $(c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c_2 = SKIP) = (c_2' = SKIP) \wedge$
 $map \ fst \ [p \leftarrow drop \ (length \ vs) \ vs_2. \ fst \ p \in S] =$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1'') \ vs_2'. \ fst \ p \in S]$
(is ?Q1' \wedge ?R2 \wedge ?Q2')
using Z and $\langle \ ?P1' \ \rangle$ by simp
hence ?Q1' and ?R2 and ?Q2' by auto
have $s_1 = t_1 (\subseteq bvars \ b)$
by (rule eq-states-while [OF AA Z], insert L N P, simp+)
hence bval b t₁
using P and Q by (blast dest: bvars-bval)

```

hence (WHILE b DO c,  $t_1, f', vs_1', ws_1'$ )  $\rightarrow$ 
  (c; WHILE b DO c,  $t_1, f', vs_1', ws_1'$ )  $\cdot\cdot$ 
moreover have (c; WHILE b DO c,  $t_1, f', vs_1', ws_1'$ )  $\rightarrow^*$ 
  ( $c_2', t_2, f', vs_2', ws_2'$ )
  using  $\langle ?Q1 \rangle$  and  $\langle ?Q1' \rangle$ 
  by (blast intro: star-seq2 star-trans)
ultimately have
  ( $c_1, t_1, f', vs_1', ws_1'$ )  $\rightarrow^*$  ( $c_2', t_2, f', vs_2', ws_2'$ )
  (is  $?R1$ )
  using P by (blast intro: star-trans)
moreover have
  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  $p \in S$ ]
  by (rule small-steps-inputs
  [OF T U  $\langle ?Q1 \rangle$   $\langle ?Q1' \rangle$   $\langle ?Q2 \rangle$   $\langle ?Q2' \rangle$ ])
ultimately have  $?R1 \wedge ?R2 \wedge ?this$ 
  using  $\langle ?R2 \rangle$  by simp
}
moreover {
  fix S
  assume
  Z:  $S \neq \{\}$  and
  AA:  $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources}$ 
    ( $\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') vs_1 s_1 f x\}$  and
  AB:  $f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags}$ 
    ( $\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') vs_1 s_1 f x \mid x. x \in S\})$ 
  have  $\forall x. \text{sources} (?cs_1' @ ?cs_2') vs_1 s_1 f x \subseteq$ 
     $\text{sources} (\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') vs_1 s_1 f x$ 
  by (blast intro: subsetD [OF sources-observe-tl])
  hence AC:  $S \subseteq \{x. s_1 = t_1$ 
    ( $\subseteq \text{sources} (?cs_1' @ ?cs_2') vs_1 s_1 f x\}$ 
  using AA by blast
  have  $\forall x. \text{sources-aux} (?cs_1' @ ?cs_2') vs_1 s_1 f x \subseteq$ 
     $\text{sources} (?cs_1' @ ?cs_2') vs_1 s_1 f x$ 
  by (blast intro: subsetD [OF sources-aux-sources])
  moreover have  $\forall x. \text{sources-aux} ?cs_1' vs_1 s_1 f x \subseteq$ 
     $\text{sources-aux} (?cs_1' @ ?cs_2') vs_1 s_1 f x$ 
  by (blast intro: subsetD [OF sources-aux-append])
  ultimately have
  AD:  $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux} ?cs_1' vs_1 s_1 f x\}$ 
  using AC by blast
  have AE:  $f = f' (\subseteq vs_1, vs_1',$ 
     $\bigcup \{\text{tags} (?cs_1' @ ?cs_2') vs_1 s_1 f x \mid x. x \in S\})$ 
  (is  $- = - (\subseteq -, -, ?T)$ )
  using AB by (simp add: tags-observe-tl)
  have
   $\bigcup \{\text{tags-aux} (?cs_1' @ ?cs_2') vs_1 s_1 f x \mid x. x \in S\} \subseteq ?T$ 
  (is  $?T' \subseteq -$ )
  by (blast intro: subsetD [OF tags-aux-tags])
}

```

moreover have
 $\bigcup \{ \text{tags-aux } ?cs_1' \text{ vs}_1 \text{ s}_1 \text{ f x} \mid x. x \in S \} \subseteq ?T'$
(is $?T'' \subseteq -$ **)**
by (*blast intro: subsetD [OF tags-aux-append]*)
ultimately have $?T'' \subseteq ?T$
by *simp*
with *AE* **have** $f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T'')$
by (*rule eq-streams-subset*)
hence *AF*: $(c, t_1, f', \text{vs}_1', \text{ws}_1') \rightarrow^*$
 $(\text{SKIP}, t_1', f', \text{vs}_1'', \text{ws}_1'')$
using *Z* **and** *AD* **and** $\langle ?P1 \rangle$ **by** *simp*
have $S \subseteq \{x. \text{s}_1' = t_1' (\subseteq \text{sources } ?cs_2' \text{ vs } \text{s}_1' \text{ f x})\}$
by (*rule sources-rhs [OF AC AE T \langle ?P2 \rangle]*)
moreover have $f = f' (\subseteq \text{vs}, \text{vs}_1'')$
 $\bigcup \{ \text{tags } ?cs_2' \text{ vs } \text{s}_1' \text{ f x} \mid x. x \in S \}$
by (*rule tags-rhs [OF AC AE T AF \langle ?P1 \rangle]*)
ultimately have $\text{s}_2 = t_2 (\subseteq S)$
using $\langle ?P2' \rangle$ **by** *blast*

}
moreover {
fix *S*
assume
 $Z: S \neq \{\}$ **and**
 $AA: S \subseteq \{x. \text{s}_1 = t_1 (\subseteq \text{sources-out}$
 $(\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x})\}$ **and**
 $AB: f = f' (\subseteq \text{vs}_1, \text{vs}_1', \bigcup \{ \text{tags-out}$
 $(\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x} \mid x. x \in S \})$
have $\forall x. \text{sources-out } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x} \subseteq$
 $\text{sources-out } (\langle \text{bvars } b \rangle \# ?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x}$
by (*blast intro: subsetD [OF sources-out-observe-tl]*)
hence *AC*: $S \subseteq \{x. \text{s}_1 = t_1$
 $(\subseteq \text{sources-out } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x})\}$
using *AA* **by** *blast*
have *AD*: $\forall x. \text{sources-aux } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x} \subseteq$
 $\text{sources-out } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x}$
by (*blast intro: subsetD [OF sources-aux-sources-out]*)
moreover have $\forall x. \text{sources-aux } ?cs_1' \text{ vs}_1 \text{ s}_1 \text{ f x} \subseteq$
 $\text{sources-aux } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x}$
by (*blast intro: subsetD [OF sources-aux-append]*)
ultimately have
 $AE: S \subseteq \{x. \text{s}_1 = t_1 (\subseteq \text{sources-aux } ?cs_1' \text{ vs}_1 \text{ s}_1 \text{ f x})\}$
using *AC* **by** *blast*
have *AF*: $f = f' (\subseteq \text{vs}_1, \text{vs}_1',$
 $\bigcup \{ \text{tags-out } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x} \mid x. x \in S \})$
(is $- = - (\subseteq -, -, ?T)$ **)**
using *AB* **by** (*simp add: tags-out-observe-tl*)
have *AG*:
 $\bigcup \{ \text{tags-aux } (?cs_1' @ ?cs_2') \text{ vs}_1 \text{ s}_1 \text{ f x} \mid x. x \in S \} \subseteq ?T$
(is $?T' \subseteq -$ **)**

by (blast intro: subsetD [OF tags-aux-tags-out])
moreover have
 $\bigcup \{tags\text{-aux } ?cs_1' vs_1 s_1 f x \mid x. x \in S\} \subseteq ?T'$
 (is $?T'' \subseteq -$)
 by (blast intro: subsetD [OF tags-aux-append])
ultimately have $?T'' \subseteq ?T$
 by simp
with AF have $f = f' (\subseteq vs_1, vs_1', ?T'')$
 by (rule eq-streams-subset)
hence AH: $(c, t_1, f', vs_1', ws_1') \rightarrow*$
 $(SKIP, t_1', f', vs_1'', ws_1'')$
 using Z and AE and $\langle ?P1 \rangle$ by simp
have AI: $S \subseteq \{x. s_1 = t_1$
 $(\subseteq sources\text{-aux } (?cs_1' @ ?cs_2') vs_1 s_1 f x)\}$
 using AC and AD by blast
have AJ: $f = f' (\subseteq vs_1, vs_1', ?T')$
 using AF and AG by (rule eq-streams-subset)
have $S \subseteq \{x. s_1' = t_1' (\subseteq sources\text{-aux } ?cs_2' vs s_1' f x)\}$
 by (rule sources-aux-rhs [OF AI AJ T $\langle ?P2 \rangle$])
moreover have $f = f' (\subseteq vs, vs_1'',$
 $\bigcup \{tags\text{-aux } ?cs_2' vs s_1' f x \mid x. x \in S\})$
 by (rule tags-aux-rhs [OF AI AJ T AH $\langle ?P1 \rangle$])
ultimately have AK:
 $(WHILE b DO c, t_1', f', vs_1'', ws_1'') \rightarrow*$
 $(c_2', t_2, f', vs_2', ws_2')$
 using Z and $\langle ?P1' \rangle$ by simp
have $\forall x. sources\text{-out } ?cs_1' vs_1 s_1 f x \subseteq$
 $sources\text{-out } (?cs_1' @ ?cs_2') vs_1 s_1 f x$
 by (blast intro: subsetD [OF sources-out-append])
hence $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-out } ?cs_1' vs_1 s_1 f x)\}$
 using AC by blast
moreover have
 $\bigcup \{tags\text{-out } ?cs_1' vs_1 s_1 f x \mid x. x \in S\} \subseteq ?T$
 (is $?T' \subseteq -$)
 by (blast intro: subsetD [OF tags-out-append])
with AF have $f = f' (\subseteq vs_1, vs_1', ?T')$
 by (rule eq-streams-subset)
ultimately have AL:
 $[p \leftarrow drop (length ws_1) ws. fst p \in S] =$
 $[p \leftarrow drop (length ws_1') ws_1''. fst p \in S]$
 using Z and $\langle ?P3 \rangle$ by simp
have $S \subseteq \{x. s_1' = t_1' (\subseteq sources\text{-out } ?cs_2' vs s_1' f x)\}$
 by (rule sources-out-rhs [OF AC AF T $\langle ?P2 \rangle$])
moreover have $f = f' (\subseteq vs, vs_1'',$
 $\bigcup \{tags\text{-out } ?cs_2' vs s_1' f x \mid x. x \in S\})$
 by (rule tags-out-rhs [OF AC AF T AH $\langle ?P1 \rangle$])
ultimately have $[p \leftarrow drop (length ws) ws_2. fst p \in S] =$
 $[p \leftarrow drop (length ws_1'') ws_2'. fst p \in S]$
 using Z and $\langle ?P3' \rangle$ by simp

hence $[p \leftarrow \text{drop} (\text{length } ws_1) ws_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop} (\text{length } ws_1') ws_2'. \text{fst } p \in S]$
by (rule *small-steps-outputs* [*OF T U AH AK AL*])
}
ultimately show
ok-flow-aux-1 $c_1 c_2 c_2' s_1 t_1 t_2 f f'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
using *R* and *V* **by** *auto*
qed
qed (*insert L, auto simp: no-upd-empty*)
qed
done
next
assume
Q: $\neg \text{bval } b \ s$ **and**
R: $\text{flow } cfs_2 = [\langle \text{bvars } b \rangle]$
(*is* $?cs = -$)
assume $(c_2, s_2, f, vs_2, ws_2) = (\text{SKIP}, s, f, vs_0, ws_0)$
hence *S*: $(c_2, s_2, f, vs_2, ws_2) = (\text{SKIP}, s_1, f, vs_1, ws_1)$
using *P* **by** *simp*
show *?thesis*
proof (rule *conjI, clarify*)
fix $t_1 f' vs_1' ws_1'$
show $\exists c_2' t_2 vs_2' ws_2'$.
ok-flow-aux-1 $c_1 c_2 c_2' s_1 t_1 t_2 f f'$
 $vs_1 vs_1' vs_2 vs_2' ws_1' ws_2' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_2 f f' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_2' ?cs$
proof (rule *exI* [*of - SKIP*], rule *exI* [*of - t₁*],
rule *exI* [*of - vs₁'*], rule *exI* [*of - ws₁'*])
{
fix *S*
assume
 $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } [\langle \text{bvars } b \rangle] vs_1 s_1 f x)\}$ **and**
 $S \neq \{\}$
hence $s_1 = t_1 (\subseteq \text{bvars } b)$
by (rule *eq-states-while, insert L N P, simp+*)
hence $\neg \text{bval } b \ t_1$
using *P* **and** *Q* **by** (*blast dest: bvars-bval*)
hence $(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (\text{SKIP}, t_1, f', vs_1', ws_1')$
using *P* **by** *simp*
}
moreover **{**
fix *S*
assume $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } [\langle \text{bvars } b \rangle] vs_1 s_1 f x)\}$
moreover **have** $\forall x. \text{sources } [] vs_1 s_1 f x \subseteq$
 $\text{sources } [\langle \text{bvars } b \rangle] vs_1 s_1 f x$

by (*blast intro!*: *sources-observe-tl*)
 ultimately have $s_1 = t_1 (\subseteq S)$
 by *auto*
 }
 ultimately show
ok-flow-aux-1 $c_1 c_2 \text{ SKIP } s_1 t_1 t_1 f f'$
 $vs_1 vs_1' vs_2 vs_1' ws_1' ws_1' ?cs \wedge$
ok-flow-aux-2 $s_1 s_2 t_1 t_1 f f' vs_1 vs_1' ?cs \wedge$
ok-flow-aux-3 $s_1 t_1 f f' vs_1 vs_1' ws_1 ws_1' ws_2 ws_1' ?cs$
 using *R* and *S* by *auto*
 qed
 qed (*insert L, auto simp: no-upd-empty*)
 qed
 next
 assume *P*: *bval* $b s$
 assume $(c;; \text{WHILE } b \text{ DO } c, s, f, vs_0, ws_0) \rightarrow^*\{tl\} cfs_1$
 $(c_1, s_1, f, vs_1, ws_1)$
 hence
 $(\exists c' cfs.$
 $c_1 = c';; \text{WHILE } b \text{ DO } c \wedge$
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c', s_1, f, vs_1, ws_1) \wedge$
 $flow (tl\ cfs_1) = flow\ cfs) \vee$
 $(\exists p\ cfs'\ cfs''.$
 $length\ cfs'' < length\ (tl\ cfs_1) \wedge$
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs'\} (\text{SKIP}, p) \wedge$
 $(\text{WHILE } b \text{ DO } c, p) \rightarrow^*\{cfs''\} (c_1, s_1, f, vs_1, ws_1) \wedge$
 $flow (tl\ cfs_1) = flow\ cfs' @ flow\ cfs'')$
 by (*rule small-steps1-seq*)
 thus ?thesis
 apply (*rule disjE*)
 apply (*erule exE*) +
 apply (*erule conjE*) +
 subgoal for $c' cfs$
 proof –
 assume
 $Q: (c, s, f, vs_0, ws_0) \rightarrow^*\{cfs\} (c', s_1, f, vs_1, ws_1)$ and
 $R: c_1 = c';; \text{WHILE } b \text{ DO } c$
 hence $(c';; \text{WHILE } b \text{ DO } c, s_1, f, vs_1, ws_1) \rightarrow^*\{cfs_2\}$
 $(c_2, s_2, f, vs_2, ws_2)$
 using *O* by *simp*
 hence
 $(\exists c'' cfs'.$
 $c_2 = c'';; \text{WHILE } b \text{ DO } c \wedge$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (c'', s_2, f, vs_2, ws_2) \wedge$
 $flow\ cfs_2 = flow\ cfs') \vee$
 $(\exists p\ cfs'\ cfs''.$
 $length\ cfs'' < length\ cfs_2 \wedge$
 $(c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (\text{SKIP}, p) \wedge$
 $(\text{WHILE } b \text{ DO } c, p) \rightarrow^*\{cfs''\} (c_2, s_2, f, vs_2, ws_2) \wedge$

$flow\ cfs_2 = flow\ cfs' @ flow\ cfs''$
by (rule *small-stepsl-seq*)
thus ?thesis
apply (rule *disjE*)
apply (erule *exE*)
apply (erule *conjE*)
subgoal for $c''\ cfs'$
proof –
assume
 $S: c_2 = c'';$ *WHILE* $b\ DO\ c$ **and**
 $T: (c', s_1, f, vs_1, ws_1) \rightarrow^*\{cfs'\} (c'', s_2, f, vs_2, ws_2)$ **and**
 $U: flow\ cfs_2 = flow\ cfs'$
(is ?cs = ?cs')
from N **have** *ok-flow-aux* { } $c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ ?cs'$
proof
assume $V: s \in Univ\ A (\subseteq state \cap X)$
have $W: s \in Univ\ B_1 (\subseteq state \cap X)$
using P **by** (*insert btyping2-approx* [*OF G V*], *simp*)
show ?thesis
by (rule B [*OF G* [*symmetric*] H [*symmetric*] I [*symmetric*]
 $L\ J\ W\ Q\ T$])
next
assume $V: s \in Univ\ C (\subseteq state \cap Y)$
have $W: s \in Univ\ B_1' (\subseteq state \cap Y)$
using P **by** (*insert btyping2-approx* [*OF I V*], *simp*)
show ?thesis
by (rule C [*OF G* [*symmetric*] H [*symmetric*] I [*symmetric*]
 $L\ K\ W\ Q\ T$])
qed
hence $V: ok-flow-aux\ \{\}\ c'\ c''\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ ?cs$
using U **by** *simp*
show ?thesis
proof (rule *conjI*, *clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
 $ok-flow-aux-1\ c'\ c''\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
 $ok-flow-aux-2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
 $ok-flow-aux-3\ s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
(is ?P1 \wedge ?P2 \wedge ?P3)
using V **by** *fastforce*
hence ?P1 **and** ?P2 **and** ?P3 **by** *auto*
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
 $ok-flow-aux-1\ c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'$
 $vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
 $ok-flow-aux-2\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
 $ok-flow-aux-3\ s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (rule *exI* [*of* - c_2']; *WHILE* $b\ DO\ c$],
rule *exI* [*of* - t_2], rule *exI* [*of* - vs_2'], rule *exI* [*of* - ws_2'])

```

{
  fix S
  assume S ≠ {} and
  S ⊆ {x. s1 = t1 (⊆ sources-aux ?cs vs1 s1 f x)} and
  f = f' (⊆ vs1, vs1')
  ∪ {tags-aux ?cs vs1 s1 f x | x. x ∈ S}
  hence
  (c1, t1, f', vs1', ws1') →*
  (c2';; WHILE b DO c, t2, f', vs2', ws2') ∧
  map fst [p←drop (length vs1) vs2. fst p ∈ S] =
  map fst [p←drop (length vs1') vs2'. fst p ∈ S]
  using R and ⟨?P1⟩ by (blast intro: star-seq2)
}
thus
ok-flow-aux-1 c1 c2 (c2';; WHILE b DO c) 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
using S and ⟨?P2⟩ and ⟨?P3⟩ by simp
qed
qed (insert L, auto simp: no-upd-empty)
qed
apply (erule exE)+
apply (erule conjE)+
subgoal for p cfs' cfs''
proof -
  assume (c', s1, f, vs1, ws1) →*{cfs'} (SKIP, p)
  moreover from this obtain s1' and vs and ws where
  S: p = (s1', f, vs, ws)
  by (blast dest: small-steps1-stream)
  ultimately have
  T: (c', s1, f, vs1, ws1) →*{cfs'} (SKIP, s1', f, vs, ws)
  by simp
  assume (WHILE b DO c, p) →*{cfs''} (c2, s2, f, vs2, ws2)
  with S have
  U: (WHILE b DO c, s1', f, vs, ws) →*{cfs''}
  (c2, s2, f, vs2, ws2)
  by simp
  assume V: flow cfs2 = flow cfs' @ flow cfs''
  (is (?cs :: flow) = ?cs1 @ ?cs2)
  from N have
  W: ok-flow-aux {} c' SKIP s1 s1' f vs1 vs ws1 ws ?cs1
proof
  assume X: s ∈ Univ A (⊆ state ∩ X)
  have Y: s ∈ Univ B1 (⊆ state ∩ X)
  using P by (insert btyping2-approx [OF G X], simp)
  show ?thesis
  by (rule B [OF G [symmetric] H [symmetric] I [symmetric]
  L J Y Q T])

```

next
assume $X: s \in Univ\ C (\subseteq state \cap Y)$
have $Y: s \in Univ\ B_1' (\subseteq state \cap Y)$
using P **by** (*insert btyping2-approx* [$OF\ I\ X$], *simp*)
show *?thesis*
by (*rule* C [$OF\ G$ [*symmetric*] H [*symmetric*] I [*symmetric*]
 $L\ K\ Y\ Q\ T$])
qed
assume $length\ cfs'' < length\ cfs_2$
hence $length\ (\ []\ @\ cfs'') < length\ (cfs_1\ @\ cfs_2)$
by *simp*
moreover **have** $\ []\ @\ cfs'' = \ []\ @\ cfs'' ..$
moreover **have**
 $(c, s, f, vs_0, ws_0) \rightarrow^*\{cfs\ @\ cfs'\}$ (*SKIP*, s_1', f, vs, ws)
using Q **and** T **by** (*rule* *small-stepsl-append*)
hence $(c, s, f, vs_0, ws_0) \Rightarrow (s_1', f, vs, ws)$
by (*auto* *dest: small-stepsl-steps simp: big-iff-small*)
hence $s_1' \in Univ\ A (\subseteq state \cap X) \cup Univ\ C (\subseteq state \cap Y)$
by (*rule* *univ-states-while* [$OF - G\ H\ I\ J\ K\ P\ N$])
moreover **have** (*WHILE* $b\ DO\ c, s_1', f, vs, ws$) $\rightarrow^*\{\ []\}$
 $(WHILE\ b\ DO\ c, s_1', f, vs, ws)$
by *simp*
ultimately **have** X :
ok-flow-aux U (*WHILE* $b\ DO\ c$) $c_2\ s_1'\ s_2\ f\ vs\ vs_2\ ws\ ws_2\ ?cs_2$
using U **by** (*rule* M [*rule-format*])
show *?thesis*
proof (*rule* *conjI, clarify*)
fix $t_1\ f'\ vs_1'\ ws_1'$
obtain c_1'' **and** t_1' **and** vs_1'' **and** ws_1'' **where**
ok-flow-aux-1 $c'\ SKIP\ c_1''\ s_1\ t_1\ t_1'\ f\ f'$
 $vs_1\ vs_1'\ vs\ vs_1''\ ws_1'\ ws_1''\ ?cs_1 \wedge$
ok-flow-aux-2 $s_1\ s_1'\ t_1\ t_1'\ f\ f'\ vs_1\ vs_1'\ ?cs_1 \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws\ ws_1''\ ?cs_1$
 $(is - \wedge\ ?P2 \wedge\ ?P3)$
using W **by** *fastforce*
hence
ok-flow-aux-1 $c'\ SKIP\ SKIP\ s_1\ t_1\ t_1'\ f\ f'$
 $vs_1\ vs_1'\ vs\ vs_1''\ ws_1'\ ws_1''\ ?cs_1$
 $(is\ ?P1)$ **and** $?P2$ **and** $?P3$ **by** *auto*
obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
ok-flow-aux-1 (*WHILE* $b\ DO\ c$) $c_2\ c_2'\ s_1'\ t_1'\ t_2\ f\ f'$
 $vs\ vs_1''\ vs_2\ vs_2'\ ws_1''\ ws_2'\ ?cs_2 \wedge$
ok-flow-aux-2 $s_1'\ s_2\ t_1'\ t_2\ f\ f'\ vs\ vs_1''\ ?cs_2 \wedge$
ok-flow-aux-3 $s_1'\ t_1'\ f\ f'\ vs\ vs_1''\ ws\ ws_1''\ ws_2\ ws_2'\ ?cs_2$
 $(is\ ?P1' \wedge\ ?P2' \wedge\ ?P3')$
using X **by** *fastforce*
hence $?P1'$ **and** $?P2'$ **and** $?P3'$ **by** *auto*
show $\exists\ c_2'\ t_2\ vs_2'\ ws_2'$.
ok-flow-aux-1 $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'$

$vs_1 \ vs_1' \ vs_2 \ vs_2' \ ws_1' \ ws_2' \ ?cs \wedge$
 $ok\text{-flow}\text{-aux}\text{-}2 \ s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ vs_1 \ vs_1' \ ?cs \wedge$
 $ok\text{-flow}\text{-aux}\text{-}3 \ s_1 \ t_1 \ f \ f' \ vs_1 \ vs_1' \ ws_1 \ ws_1' \ ws_2 \ ws_2' \ ?cs$
proof (rule exI [of - c_2], rule exI [of - t_2],
rule exI [of - vs_2], rule exI [of - ws_2])
{
fix S
assume
 $Y: S \neq \{\}$ **and**
 $Z: S \subseteq \{x. s_1 = t_1$
 $(\subseteq sources\text{-aux} \ (?cs_1 \ @ \ ?cs_2) \ vs_1 \ s_1 \ f \ x)\}$ **and**
 $AA: f = f' (\subseteq vs_1, vs_1',$
 $\bigcup \{tags\text{-aux} \ (?cs_1 \ @ \ ?cs_2) \ vs_1 \ s_1 \ f \ x \mid x. x \in S\}$
 $(is \ - = - (\subseteq -, -, ?T))$
have $\forall x. sources\text{-aux} \ ?cs_1 \ vs_1 \ s_1 \ f \ x \subseteq$
 $sources\text{-aux} \ (?cs_1 \ @ \ ?cs_2) \ vs_1 \ s_1 \ f \ x$
by (blast intro: subsetD [OF sources-aux-append])
hence $S \subseteq \{x. s_1 = t_1 (\subseteq sources\text{-aux} \ ?cs_1 \ vs_1 \ s_1 \ f \ x)\}$
using Z **by** blast
moreover have
 $\bigcup \{tags\text{-aux} \ ?cs_1 \ vs_1 \ s_1 \ f \ x \mid x. x \in S\} \subseteq ?T$
 $(is \ ?T' \subseteq -)$
by (blast intro: subsetD [OF tags-aux-append])
with AA **have** $f = f' (\subseteq vs_1, vs_1', ?T')$
by (rule eq-streams-subset)
ultimately have
 $(c', t_1, f', vs_1', ws_1') \rightarrow^*$
 $(SKIP, t_1', f', vs_1'', ws_1'') \wedge$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1) \ vs. \ fst \ p \in S] =$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1') \ vs_1''. \ fst \ p \in S]$
 $(is \ ?Q1 \wedge ?Q2)$
using Y **and** $\langle ?P1 \rangle$ **by** simp
hence $?Q1$ **and** $?Q2$ **by** auto
have $S \subseteq \{x. s_1' = t_1' (\subseteq sources\text{-aux} \ ?cs_2 \ vs \ s_1' \ f \ x)\}$
by (rule sources-aux-rhs [OF $Z \ AA \ T \ \langle ?P2 \rangle$])
moreover have $f = f' (\subseteq vs, vs_1'',$
 $\bigcup \{tags\text{-aux} \ ?cs_2 \ vs \ s_1' \ f \ x \mid x. x \in S\})$
by (rule tags-aux-rhs [OF $Z \ AA \ T \ \langle ?Q1 \rangle \ \langle ?P1 \rangle$])
ultimately have
 $(WHILE \ b \ DO \ c, t_1', f', vs_1'', ws_1'') \rightarrow^*$
 $(c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c_2 = SKIP) = (c_2' = SKIP) \wedge$
 $map \ fst \ [p \leftarrow drop \ (length \ vs) \ vs_2. \ fst \ p \in S] =$
 $map \ fst \ [p \leftarrow drop \ (length \ vs_1'') \ vs_2'. \ fst \ p \in S]$
 $(is \ ?Q1' \wedge ?R2 \wedge ?Q2')$
using Y **and** $\langle ?P1' \rangle$ **by** simp
hence $?Q1'$ **and** $?R2$ **and** $?Q2'$ **by** auto
from R **and** $\langle ?Q1 \rangle$ **and** $\langle ?Q1' \rangle$ **have**
 $(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2')$

```

    (is ?R1)
    by (blast intro: star-seq2 star-trans)
  moreover have
    map fst [p←drop (length vs1) vs2. fst p ∈ S] =
      map fst [p←drop (length vs1') vs2'. fst p ∈ S]
    by (rule small-steps-inputs
        [OF T U ‹?Q1› ‹?Q1'› ‹?Q2› ‹?Q2'›])
  ultimately have ?R1 ∧ ?R2 ∧ ?this
    using ‹?R2› by simp
}
moreover {
  fix S
  assume
    Y: S ≠ {} and
    Z: S ⊆ {x. s1 = t1
      (⊆ sources (?cs1 @ ?cs2) vs1 s1 f x)} and
    AA: f = f' (⊆ vs1, vs1',
      ⋃ {tags (?cs1 @ ?cs2) vs1 s1 f x | x. x ∈ S})
      (is - = - (⊆ -, -, ?T))
  have ∀ x. sources-aux (?cs1 @ ?cs2) vs1 s1 f x ⊆
    sources (?cs1 @ ?cs2) vs1 s1 f x
    by (blast intro: subsetD [OF sources-aux-sources])
  moreover have ∀ x. sources-aux ?cs1 vs1 s1 f x ⊆
    sources-aux (?cs1 @ ?cs2) vs1 s1 f x
    by (blast intro: subsetD [OF sources-aux-append])
  ultimately have
    AB: S ⊆ {x. s1 = t1 (⊆ sources-aux ?cs1 vs1 s1 f x)}
    using Z by blast
  have
    ⋃ {tags-aux (?cs1 @ ?cs2) vs1 s1 f x | x. x ∈ S} ⊆ ?T
    (is ?T' ⊆ -)
    by (blast intro: subsetD [OF tags-aux-tags])
  moreover have
    ⋃ {tags-aux ?cs1 vs1 s1 f x | x. x ∈ S} ⊆ ?T'
    (is ?T'' ⊆ -)
    by (blast intro: subsetD [OF tags-aux-append])
  ultimately have ?T'' ⊆ ?T
    by simp
  with AA have f = f' (⊆ vs1, vs1', ?T'')
    by (rule eq-streams-subset)
  hence AC: (c', t1, f', vs1', ws1') →*
    (SKIP, t1', f', vs1'', ws1'')
    using Y and AB and ‹?P1› by simp
  have S ⊆ {x. s1' = t1' (⊆ sources ?cs2 vs s1' f x)}
    by (rule sources-rhs [OF Z AA T ‹?P2›])
  moreover have f = f' (⊆ vs, vs1'',
    ⋃ {tags ?cs2 vs s1' f x | x. x ∈ S})
    by (rule tags-rhs [OF Z AA T AC ‹?P1›])
  ultimately have s2 = t2 (⊆ S)

```

```

    using ⟨?P2'⟩ by blast
  }
  moreover {
    fix S
    assume
      Y: S ≠ {} and
      Z: S ⊆ {x. s1 = t1}
        (⊆ sources-out (?cs1 @ ?cs2) vs1 s1 f x)} and
      AA: f = f' (⊆ vs1, vs1',
        ⋃ {tags-out (?cs1 @ ?cs2) vs1 s1 f x | x. x ∈ S})
        (is - = - (⊆ -, -, ?T))
    have AB: ∀ x. sources-aux (?cs1 @ ?cs2) vs1 s1 f x ⊆
      sources-out (?cs1 @ ?cs2) vs1 s1 f x
      by (blast intro: subsetD [OF sources-aux-sources-out])
    moreover have ∀ x. sources-aux ?cs1 vs1 s1 f x ⊆
      sources-aux (?cs1 @ ?cs2) vs1 s1 f x
      by (blast intro: subsetD [OF sources-aux-append])
    ultimately have
      AC: S ⊆ {x. s1 = t1} (⊆ sources-aux ?cs1 vs1 s1 f x)
      using Z by blast
    have AD:
      ⋃ {tags-aux (?cs1 @ ?cs2) vs1 s1 f x | x. x ∈ S} ⊆ ?T
      (is ?T' ⊆ -)
      by (blast intro: subsetD [OF tags-aux-tags-out])
    moreover have
      ⋃ {tags-aux ?cs1 vs1 s1 f x | x. x ∈ S} ⊆ ?T'
      (is ?T'' ⊆ -)
      by (blast intro: subsetD [OF tags-aux-append])
    ultimately have ?T'' ⊆ ?T
      by simp
    with AA have f = f' (⊆ vs1, vs1', ?T'')
      by (rule eq-streams-subset)
    hence AE: (c', t1, f', vs1', ws1') →*
      (SKIP, t1', f', vs1'', ws1'')
      using Y and AC and ⟨?P1'⟩ by simp
    have AF: S ⊆ {x. s1 = t1}
      (⊆ sources-aux (?cs1 @ ?cs2) vs1 s1 f x)
      using Z and AB by blast
    have AG: f = f' (⊆ vs1, vs1', ?T')
      using AA and AD by (rule eq-streams-subset)
    have S ⊆ {x. s1' = t1'} (⊆ sources-aux ?cs2 vs s1' f x)
      by (rule sources-aux-rhs [OF AF AG T ⟨?P2'⟩])
    moreover have f = f' (⊆ vs, vs1'',
      ⋃ {tags-aux ?cs2 vs s1' f x | x. x ∈ S})
      by (rule tags-aux-rhs [OF AF AG T AE ⟨?P1'⟩])
    ultimately have
      AH: (WHILE b DO c, t1', f', vs1'', ws1'') →*
        (c2', t2', f', vs2', ws2')
      using Y and ⟨?P1'⟩ by simp
  }

```

have $\forall x. \text{sources-out } ?cs_1 \text{ vs}_1 \text{ s}_1 \text{ f } x \subseteq$
 $\text{sources-out } (?cs_1 @ ?cs_2) \text{ vs}_1 \text{ s}_1 \text{ f } x$
by (*blast intro: subsetD [OF sources-out-append]*)
hence $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } ?cs_1 \text{ vs}_1 \text{ s}_1 \text{ f } x)\}$
using Z **by** *blast*
moreover have
 $\bigcup \{\text{tags-out } ?cs_1 \text{ vs}_1 \text{ s}_1 \text{ f } x \mid x. x \in S\} \subseteq ?T$
(is $?T' \subseteq -$ **)**
by (*blast intro: subsetD [OF tags-out-append]*)
with AA **have** $f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T')$
by (*rule eq-streams-subset*)
ultimately have $AI:$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1) \text{ ws}. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1') \text{ ws}_1''. \text{fst } p \in S]$
using Y **and** $\langle ?P3 \rangle$ **by** *simp*
have $S \subseteq \{x. s_1' = t_1' (\subseteq \text{sources-out } ?cs_2 \text{ vs } s_1' \text{ f } x)\}$
by (*rule sources-out-rhs [OF Z AA T \langle ?P2 \rangle]*)
moreover have $f = f' (\subseteq \text{vs}, \text{vs}_1'')$
 $\bigcup \{\text{tags-out } ?cs_2 \text{ vs } s_1' \text{ f } x \mid x. x \in S\}$
by (*rule tags-out-rhs [OF Z AA T AE \langle ?P1 \rangle]*)
ultimately have $[p \leftarrow \text{drop } (\text{length } \text{ws}) \text{ ws}_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1'') \text{ ws}_2'. \text{fst } p \in S]$
using Y **and** $\langle ?P3 \rangle$ **by** *simp*
hence $[p \leftarrow \text{drop } (\text{length } \text{ws}_1) \text{ ws}_2. \text{fst } p \in S] =$
 $[p \leftarrow \text{drop } (\text{length } \text{ws}_1') \text{ ws}_2'. \text{fst } p \in S]$
by (*rule small-steps-outputs [OF T U AE AH AI]*)
}
ultimately show
 $ok\text{-flow-aux-1 } c_1 \text{ } c_2 \text{ } c_2' \text{ } s_1 \text{ } t_1 \text{ } t_2 \text{ } f \text{ } f'$
 $\text{vs}_1 \text{ vs}_1' \text{ vs}_2 \text{ vs}_2' \text{ ws}_1' \text{ ws}_2' \text{ ?cs} \wedge$
 $ok\text{-flow-aux-2 } s_1 \text{ } s_2 \text{ } t_1 \text{ } t_2 \text{ } f \text{ } f' \text{ vs}_1 \text{ vs}_1' \text{ ?cs} \wedge$
 $ok\text{-flow-aux-3 } s_1 \text{ } t_1 \text{ } f \text{ } f' \text{ vs}_1 \text{ vs}_1' \text{ ws}_1 \text{ ws}_1' \text{ ws}_2 \text{ ws}_2' \text{ ?cs}$
using V **by** *auto*
qed
qed (*insert L, auto simp: no-upd-empty*)
qed
done
qed
apply (*erule exE*)
apply (*erule conjE*)
subgoal for $p \text{ cfs}' \text{ cfs}''$
proof –
assume $(c, s, f, \text{vs}_0, \text{ws}_0) \rightarrow^* \{cfs'\}$ (*SKIP*, p)
moreover from this obtain s_1' **and** vs **and** ws **where**
 $Q: p = (s_1', f, \text{vs}, \text{ws})$
by (*blast dest: small-stepsl-stream*)
ultimately have
 $R: (c, s, f, \text{vs}_0, \text{ws}_0) \rightarrow^* \{cfs'\}$ (*SKIP*, $s_1', f, \text{vs}, \text{ws}$)
by *simp*

```

assume (WHILE b DO c, p)  $\rightarrow^*\{cfs''\}$  (c1, s1, f, vs1, ws1)
with Q have S:
  (WHILE b DO c, s'1, f, vs, ws)  $\rightarrow^*\{cfs''\}$  (c1, s1, f, vs1, ws1)
  by simp
assume length cfs'' < length (tl cfs1)
hence length (cfs'' @ cfs2) < length (cfs1 @ cfs2)
  by simp
moreover have cfs'' @ cfs2 = cfs'' @ cfs2 ..
moreover have (c, s, f, vs0, ws0)  $\Rightarrow$  (s'1, f, vs, ws)
  using R by (auto dest: small-stepsl-steps simp: big-iff-small)
hence s'1  $\in$  Univ A ( $\subseteq$  state  $\cap$  X)  $\cup$  Univ C ( $\subseteq$  state  $\cap$  Y)
  by (rule univ-states-while [OF - G H I J K P N])
ultimately show ?thesis
  using S and O by (rule M [rule-format])
qed
done
next
assume (c1, s1, f, vs1, ws1) = (SKIP, s, f, vs0, ws0)
moreover from this have
  (c2, s2, f, vs2, ws2) = (SKIP, s1, f, vs1, ws1)  $\wedge$  flow cfs2 = []
  using O by (blast intro!: small-stepsl-skip)
ultimately show ?thesis
  by (insert L, fastforce)
qed
qed
qed

```

lemma *ctyping2-correct-aux*:

```

 $\llbracket (U, v) \models c (\subseteq A, X) = \text{Some } (B, Y); s \in \text{Univ } A (\subseteq \text{state} \cap X);$ 
  (c, s, f, vs0, ws0)  $\rightarrow^*\{cfs_1\}$  (c1, s1, f, vs1, ws1);
  (c1, s1, f, vs1, ws1)  $\rightarrow^*\{cfs_2\}$  (c2, s2, f, vs2, ws2)  $\implies$ 
ok-flow-aux U c1 c2 s1 s2 f vs1 vs2 ws1 ws2 (flow cfs2)
apply (induction (U, v) c A X arbitrary: B Y U v c1 c2 s1 s2
  vs0 vs1 vs2 ws0 ws1 ws2 cfs1 cfs2 rule: ctyping2.induct)
  apply fastforce
  apply (erule ctyping2-correct-aux-assign, assumption+)
  apply (erule ctyping2-correct-aux-input, assumption+)
  apply (erule ctyping2-correct-aux-output, assumption+)
  apply (erule ctyping2-correct-aux-seq, assumption+)
  apply (erule ctyping2-correct-aux-or, assumption+)
  apply (erule ctyping2-correct-aux-if, assumption+)
  apply (erule ctyping2-correct-aux-while, assumption+)
done

```

theorem *ctyping2-correct*:

```

assumes A: (U, v)  $\models c (\subseteq A, X) = \text{Some } (B, Y)$ 
shows correct c A X
proof (subst correct-def, clarify)

```

fix $s\ t\ c_1\ c_2\ s_1\ s_2\ f\ vs\ vs_1\ vs_2\ ws\ ws_1\ ws_2\ cfs_2\ t_1\ f'\ vs_1'\ ws_1'$
let $?cs = \text{flow } cfs_2$
assume $t \in A$ **and** $s = t (\subseteq \text{state} \cap X)$
hence $s \in \text{Univ } A (\subseteq \text{state} \cap X)$
by *blast*
moreover assume $(c, s, f, vs, ws) \rightarrow^* (c_1, s_1, f, vs_1, ws_1)$
then obtain cfs_1 **where** $(c, s, f, vs, ws) \rightarrow^* \{cfs_1\} (c_1, s_1, f, vs_1, ws_1)$
by (*blast dest: small-steps-stepsl*)
moreover assume $(c_1, s_1, f, vs_1, ws_1) \rightarrow^* \{cfs_2\} (c_2, s_2, f, vs_2, ws_2)$
ultimately have *ok-flow-aux* $U\ c_1\ c_2\ s_1\ s_2\ f\ vs_1\ vs_2\ ws_1\ ws_2\ ?cs$
by (*rule ctyping2-correct-aux [OF A]*)
then obtain c_2' **and** t_2 **and** vs_2' **and** ws_2' **where**
ok-flow-aux-1 $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
ok-flow-aux-2 $s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ ?cs \wedge$
ok-flow-aux-3 $s_1\ t_1\ f\ f'\ vs_1\ vs_1'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
(is $?P1 \wedge ?P2 \wedge ?P3$ **)**
by *fastforce*
hence $?P1$ **and** $?P2$ **and** $?P3$ **by** *auto*
show $\exists c_2'\ t_2\ vs_2'\ ws_2'$.
ok-flow-1 $c_1\ c_2\ c_2'\ s_1\ s_2\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1'\ ws_2'\ ?cs \wedge$
ok-flow-2 $c_1\ c_2\ c_2'\ s_1\ t_1\ t_2\ f\ f'\ vs_1\ vs_1'\ vs_2\ vs_2'\ ws_1\ ws_1'\ ws_2\ ws_2'\ ?cs$
proof (*rule exI [of - c_2']*, *rule exI [of - t_2]*,
rule exI [of - vs_2'], *rule exI [of - ws_2']*)
{
fix S
assume
 $B: S \neq \{\}$ **and**
 $C: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources } ?cs\ vs_1\ s_1\ f\ x)\}$ **and**
 $D: f = f' (\subseteq vs_1, vs_1', \bigcup \{\text{tags } ?cs\ vs_1\ s_1\ f\ x \mid x. x \in S\})$
(is $- = - (\subseteq -, -, ?T)$ **)**
have $\forall x. \text{sources-aux } ?cs\ vs_1\ s_1\ f\ x \subseteq \text{sources } ?cs\ vs_1\ s_1\ f\ x$
by (*blast intro: subsetD [OF sources-aux-sources]*)
hence $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs\ vs_1\ s_1\ f\ x)\}$
using C **by** *blast*
moreover have $\bigcup \{\text{tags-aux } ?cs\ vs_1\ s_1\ f\ x \mid x. x \in S\} \subseteq ?T$
(is $?T' \subseteq -$ **)**
by (*blast intro: subsetD [OF tags-aux-tags]*)
with D **have** $f = f' (\subseteq vs_1, vs_1', ?T')$
by (*rule eq-streams-subset*)
ultimately have
 $(c_1, t_1, f', vs_1', ws_1') \rightarrow^* (c_2', t_2, f', vs_2', ws_2') \wedge$
 $(c_2 = \text{SKIP}) = (c_2' = \text{SKIP}) \wedge$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1)\ vs_2. \text{fst } p \in S] =$
 $\text{map fst } [p \leftarrow \text{drop } (\text{length } vs_1')\ vs_2'. \text{fst } p \in S]$
(is $?Q$ **)**
using B **and** $\langle ?P1 \rangle$ **by** *simp*
moreover have $s_2 = t_2 (\subseteq S)$
using B **and** C **and** D **and** $\langle ?P2 \rangle$ **by** *simp*
ultimately have $?Q \wedge ?this ..$

```

}
moreover {
  fix  $S$ 
  assume
     $B: S \neq \{\}$  and
     $C: S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-out } ?cs \text{ vs}_1 \text{ s}_1 f x)\}$  and
     $D: f = f' (\subseteq \text{vs}_1, \text{vs}_1', \cup \{\text{tags-out } ?cs \text{ vs}_1 \text{ s}_1 f x \mid x. x \in S\})$ 
      (is  $- = - (\subseteq -, -, ?T)$ )
  have  $\forall x. \text{sources-aux } ?cs \text{ vs}_1 \text{ s}_1 f x \subseteq \text{sources-out } ?cs \text{ vs}_1 \text{ s}_1 f x$ 
    by (blast intro: subsetD [OF sources-aux-sources-out])
  hence  $S \subseteq \{x. s_1 = t_1 (\subseteq \text{sources-aux } ?cs \text{ vs}_1 \text{ s}_1 f x)\}$ 
    using  $C$  by blast
  moreover have  $\cup \{\text{tags-aux } ?cs \text{ vs}_1 \text{ s}_1 f x \mid x. x \in S\} \subseteq ?T$ 
    (is  $?T' \subseteq -$ )
    by (blast intro: subsetD [OF tags-aux-tags-out])
  with  $D$  have  $f = f' (\subseteq \text{vs}_1, \text{vs}_1', ?T')$ 
    by (rule eq-streams-subset)
  ultimately have
     $(c_1, t_1, f', \text{vs}_1', \text{ws}_1') \rightarrow^* (c_2', t_2, f', \text{vs}_2', \text{ws}_2') \wedge$ 
     $(c_2 = \text{SKIP}) = (c_2' = \text{SKIP}) \wedge$ 
     $\text{map fst } [p \leftarrow \text{drop } (\text{length } \text{vs}_1) \text{ vs}_2. \text{fst } p \in S] =$ 
     $\text{map fst } [p \leftarrow \text{drop } (\text{length } \text{vs}_1') \text{ vs}_2'. \text{fst } p \in S]$ 
    (is  $?Q$ )
    using  $B$  and  $\langle ?P1 \rangle$  by simp
  moreover have
     $[p \leftarrow \text{drop } (\text{length } \text{ws}_1) \text{ ws}_2. \text{fst } p \in S] =$ 
     $[p \leftarrow \text{drop } (\text{length } \text{ws}_1') \text{ ws}_2'. \text{fst } p \in S]$ 
    using  $B$  and  $C$  and  $D$  and  $\langle ?P3 \rangle$  by simp
  ultimately have  $?Q \wedge ?this ..$ 
}
ultimately show
  ok-flow-1  $c_1 \ c_2 \ c_2' \ s_1 \ s_2 \ t_1 \ t_2 \ f \ f' \ \text{vs}_1 \ \text{vs}_1' \ \text{vs}_2 \ \text{vs}_2' \ \text{ws}_1' \ \text{ws}_2' \ ?cs \wedge$ 
  ok-flow-2  $c_1 \ c_2 \ c_2' \ s_1 \ t_1 \ t_2 \ f \ f' \ \text{vs}_1 \ \text{vs}_1' \ \text{vs}_2 \ \text{vs}_2' \ \text{ws}_1 \ \text{ws}_1' \ \text{ws}_2 \ \text{ws}_2' \ ?cs$ 
  by auto
qed
qed
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.