# A Logic for Information Flow Analysis with an Application to Forward Slicing of Simple Imperative Programs

Torben Amtoft and Anindya Banerjee [a,1,2]

[a] *Department of Computing and Information Sciences*
*Kansas State University, Manhattan KS 66506, USA*

**Abstract**

We specify an information flow analysis for a simple imperative language, using a Hoare-like logic. The logic facilitates static checking of a larger class of programs than can be checked by extant type-based approaches in which a program is deemed insecure when it contains an insecure subprogram. The logic is based on an abstract interpretation of program traces that makes independence between program variables explicit. Unlike other, more precise, approaches based on Hoare logics, our approach does not require a theorem prover to generate invariants. We demonstrate the modularity of our approach by showing that a frame rule holds in our logic. Finally, we show how our logic can be applied to a program transformation, namely, forward slicing: given a derivation of a program in the logic, with the information that variable `l` is independent of variable `h`, the slicing transformation systematically creates the forward `l`-slice of the program: the slice contains all the commands independent of `h`. We show that the slicing transformation is semantics preserving.

*Key words:* Abstract interpretation, denotational semantics, frame rule, Hoare logic, information flow analysis, program slicing, strongest postcondition.

## 1 Introduction

This paper specifies an information flow analysis using a Hoare-like logic and considers an application of the logic to forward slicing in simple imperative

*Email address:* {tamtoft,ab}@cis.ksu.edu (Torben Amtoft and Anindya Banerjee).

[1] Supported by NSF grants CCR-0296182 and CCR-0209205
[2] Supported by NSF grants CCR-0296182 and CCR-0209205 and ITR-0326577

programs.

Given a system with high, or secret (H), and low, or public (L) inputs and outputs, where $\mathsf{L} \leq \mathsf{H}$ is a security lattice, a classic security problem is how to enforce the following end-to-end *confidentiality* policy: protect secret data, i.e., prevent leaks of secrets at public output channels. An information flow analysis checks if a program satisfies the policy. Denning and Denning were the first to formulate an information flow analysis for confidentiality[13]. Subsequent advances have been comprehensively summarized in the recent survey by Sabelfeld and Myers [28]. An oft-used approach for specifying static analyses for information flow is *security type systems* [24,31]. Security types can be assigned to program variables and expressions annotated with security levels. Security typing rules prevent leaks of secret information to public channels. For example, the security typing rule for assignment prevents H data from being assigned to a L variable. A well-typed program "protects secrets", i.e., no information flows from H to L during program execution.

In the security literature, "protects secrets" is formalized as *noninterference* [15] and is described in terms of an "indistinguishability" relation on states. Two program states are indistinguishable for L if they agree on values of L variables. The noninterference property says that any two runs of a program starting from two initial states that are indistinguishable for L, yield two final states that are indistinguishable for L. The two initial states may differ on values of H variables but not on values of L variables; the two final states must agree on the current values of L variables. One reading of the noninterference property is as a form of independence [9]: L output is independent of H inputs. It is this notion of independence that is made explicit in the information flow analysis specified in this paper.

A shortcoming of usual type-based approaches for information flow [5,16,31,25] is that a type system can be too imprecise. Consider the sequential program $\mathsf{l} := \mathsf{h}\,;\mathsf{l} := 0$, where $\mathsf{l}$ has type L and $\mathsf{h}$ has type H. Although a programmer would never write such a program, it may arise naturally as a result of program transformation. The program is rejected by a security type system on account of the first assignment. But the program obviously satisfies noninterference – final states of any two runs of the program will always have the same value, 0, for $\mathsf{l}$ and are thus indistinguishable for L. Similarly, the program $\mathsf{h} := \mathsf{l}\,;\mathsf{l} := \mathsf{h}$ is secure, yet is rejected by a security type system.

How can we admit such programs? Our inspiration comes from abstract interpretation [10], which can be viewed as a method for statically computing approximations of program invariants [11]. A benefit of this view is that the static abstraction of a program invariant can be used to annotate a program with pre- and postconditions and the annotated program can be checked against a Hoare-like logic. In information flow analysis, the invariant of inter-

est is *independence*, for which we use the notation $[x \ltimes w]$ to denote that $x$ is independent of variable $w$. The intuition is this: a command denotes a *trace transformer* (formalized in Section 2) that transforms a pre-trace to a post-trace. A trace, in turn, is a store transformer, that transforms an initial store to either a current store or $\perp$ where $\perp$ denotes nontermination. If $x$ is a variable, then $[x \ltimes w]$ holds for a trace $T$ provided for any two initial stores that differ only on the value of $w$, if $T$ transforms them into current stores that are non-$\perp$ then the current stores agree on the value of $x$. Alternatively, if $x$ is $\perp$, then $[\perp \ltimes w]$ holds for $T$ provided for any two initial stores that differ only on the value of $w$, if $T$ transforms them into two current stores then the one store is $\perp$ if and only if the other store is. The intuition above is just a convenient restatement of noninterference but we tie it to the static notion of independence.

Our approach statically computes finite abstractions of the concrete traces before and after the execution of a command. The notation $T^{\#}$ will be used to describe an abstraction of a concrete trace $T$. This is formalized in Section 3. We formulate (in Section 4) a Hoare-like logic for checking independences and show (Section 5) that a checked program satisfies noninterference. The assertion language of the logic is decidable since it is just the language of finite sets of independences with subset inclusion. Specifications in the logic have the form $\{T_0^{\#}\}\ C\ \{T^{\#}\}$. Given precondition $T^{\#}$, we show in Section 7 how to compute strongest postconditions; for programs with loops, this necessitates a fixpoint computation[3]. We show that the logic deems the program $l := h; l := 0$ secure: the strongest postcondition of the program contains the independence $[l \ltimes h]$.

Our approach falls in between type-based analysis and full verification where verification conditions for loops depend on loop invariants generated by a theorem prover. Instead, we approximate invariants using a fixpoint computation. Our approach is modular and we show that our logic satisfies a frame rule (Section 8). The frame rule permits local reasoning about a program: the relevant independences for a program are only those $[x \ltimes w]$ where $x$ occurs in the program. Moreover, in a larger context, the frame rule allows the following inference (in analogy with [22]): start with a specification $\{T_0^{\#}\}\ C\ \{T^{\#}\}$ describing independences before and after store modifications; then, $\{T_0^{\#} \cup T_1^{\#}\}\ C\ \{T^{\#} \cup T_1^{\#}\}$ holds provided $C$ does not modify any variable $y$ where $[y \ltimes w]$ appears in $T_1^{\#}$. The initial specification $\{T_0^{\#}\}\ C\ \{T^{\#}\}$, can reason with only the slice of store that $C$ touches.

[3] The set of independences is a finite lattice, hence the fixpoint computation will terminate.

**Contributions.** To summarize, this paper makes three contributions. First and foremost, we formulate information flow analysis in a logical form via a Hoare-like logic. The approach deems more programs secure than extant type-based approaches. In Section 9, we show how our logic conservatively extends the security type system of Smith and Volpano [31], by showing that any well-typed program in their system satisfies the invariant $[\mathtt{l} \ltimes \mathtt{h}]$. Secondly, we describe the relationship between information flow and program dependence, explored in [1,17], in a more direct manner by computing independences between program variables. The independences themselves are static descriptions of the noninterference property. The development in this paper considers *nontermination sensitive* noninterference: we assume that an attacker can observe nontermination. Finally, in Section 6, we show an application of our logic to forward slicing. Given a derivation of a program in the logic, with the information that variable $\mathtt{l}$ is independent of variable $\mathtt{h}$, the slicing transformation systematically creates the forward $\mathtt{l}$-slice of the program: the slice contains all the commands independent of $\mathtt{h}$. We show that the slicing transformation is semantics preserving.

**Differences from the conference version.** Apart from the removal of some infelicities of notation, we have made three additional contributions not present in the conference version of this paper [2].

- We consider nontermination sensitive noninterference here, compared to nontermination insensitive noninterference in the conference version. For that purpose, we needed to add a new kind of independence, which in turn necessitated changes in the logic – specifically, in the rule for loops.
- To prove the resulting logic semantically correct, the semantics had to be modified, since the one given in [2] had no explicit notion of non-termination. As an extra benefit, the resulting denotational semantics is significantly simpler than the previous one.
- We show an application of the logic to forward slicing. Although the connection between information flow analysis and slicing was explored by Abadi et. al. [1], that paper did not provide a means to compute forward slices, which we present here.

On the other hand, the conference version of this paper contained a section on counterexample generation, which we have chosen to omit in this paper. We feel that the results might merit a separate paper after some strengthening.

## 2 Language: Syntax, Traces, Semantics

This section gives the syntax of a simple imperative language, formalizes the notion of traces, and gives the semantics of the language in terms of traces.

**Syntax.** We consider a simple imperative language with assignment, sequencing, conditionals and loops as formalized by the following BNF. Commands $\mathsf{C} \in \mathbf{Cmd}$ are given by the syntax

$$\mathsf{C} ::= \mathsf{x} := \mathsf{E} \mid \mathsf{C}_1 \,;\, \mathsf{C}_2 \mid \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2 \mid \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}$$

where $\mathbf{Var}$ is an infinite set of variables, $\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{w} \in \mathbf{Var}$ range over variables and where $\mathsf{E} \in \mathbf{Exp}$ ranges over expressions. Expressions are left unspecified but we shall assume the existence of a function fv(E) that computes the free variables of expression $\mathsf{E}$. For commands, fv(C) is defined in the obvious way. We also define a function $\mathit{modified} : \mathbf{Cmd} \to \mathcal{P}(\mathbf{Var})$ that given a command, returns the set of variables potentially assigned to by the command.

$$\begin{aligned}
\mathit{modified}(\mathsf{x} := \mathsf{E}) &= \{\mathsf{x}\} \\
\mathit{modified}(\mathsf{C}_1 \,;\, \mathsf{C}_2) &= \mathit{modified}(\mathsf{C}_1) \cup \mathit{modified}(\mathsf{C}_2) \\
\mathit{modified}(\textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2) &= \mathit{modified}(\mathsf{C}_1) \cup \mathit{modified}(\mathsf{C}_2) \\
\mathit{modified}(\textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}) &= \mathit{modified}(\mathsf{C})
\end{aligned}$$

In our examples, we shall often use a **skip** command; the formal treatment of that command (wrt. the semantics and the logic) should be obvious and we shall not bother to write down the rules explicitly.

**Stores.** A store, $\mathsf{s} \in \mathbf{Sto}$, associates each variable with its current value; here values $\mathsf{v} \in \mathbf{Val}$ are yet unspecified but we assume that there exists a predicate *true?* on $\mathbf{Val}$. (For instance, we could have $\mathbf{Val}$ as the set of integers and let *true?*$(\mathsf{v})$ be defined as $\mathsf{v} \neq 0$). The following notation is used to denote a store update:

- $[\mathsf{s} \mid \mathsf{y} \mapsto \mathsf{v}]$ returns a store $\mathsf{s}'$ with the property: for all $\mathsf{x} \in \mathbf{Var}$, if $\mathsf{x} \neq \mathsf{y}$ then $\mathsf{s}'\,\mathsf{x} = \mathsf{s}\,\mathsf{x}$; but $\mathsf{s}'(\mathsf{y}) = \mathsf{v}$.

Stores $\mathsf{s}_1, \mathsf{s}_2 \in \mathbf{Sto}_\perp$ agree on $\mathsf{x} \in \mathbf{Var} \cup \{\perp\}$, written, $\mathsf{s}_1 \overset{\mathsf{x}}{=} \mathsf{s}_2$, when either of the following conditions hold: (a) $\mathsf{s}_1, \mathsf{s}_2, \mathsf{x}$ are all non-$\perp$ and then $\mathsf{s}_1\,\mathsf{x} = \mathsf{s}_2\,\mathsf{x}$; (b) $\mathsf{x} \neq \perp$ and either $\mathsf{s}_1 = \perp$ or $\mathsf{s}_2 = \perp$ (or both); (c) $\mathsf{x} = \perp$ and then $\mathsf{s}_1 = \perp$ if and only if $\mathsf{s}_2 = \perp$. This is made precise in Fig. 1 and motivated shortly,

For partial correctness – i.e., when we do not care about nontermination – we define, for $x \neq \bot$:

$$s_1 \stackrel{x}{=} s_2 \iff ((s_1 \neq \bot \wedge s_2 \neq \bot) \Rightarrow s_1\, x = s_2\, x)$$

For total correctness – i.e., when we are interested in nontermination – we additionally define, for $x = \bot$:

$$s_1 \stackrel{x}{=} s_2 \iff (s_1 = \bot \iff s_2 = \bot)$$

Fig. 1. When are two stores equal on $x$?

in Section 3. Note that for $s_1, s_2 \in \mathbf{Sto}$ and $x \in \mathbf{Var}$, $s_1 \stackrel{x}{=} s_2$ amounts to $s_1\, x = s_2\, x$.

We write $s_1 \underset{w}{=} s_2$, where $s_1, s_2 \in \mathbf{Sto}$ and $w \in \mathbf{Var}$, to denote that for all variables $y \neq w$, $s_1\, y = s_2\, y$ holds. That is, the values of all variables, *except of* $w$, are equal in stores $s_1$ and $s_2$.

**Traces.** A trace, $T \in \mathbf{Trc}$, maps an initial store to either a current store or $\bot$, where $\bot$ denotes nontermination. Thus $\mathbf{Trc} = \mathbf{Sto} \to \mathbf{Sto}_\bot$. Note that $\mathbf{Sto}_\bot$ is a complete partial order (CPO) with the ordering $\sqsubseteq$ defined as: $s_1 \sqsubseteq s_2$ iff either $s_1 = \bot$ or $s_1$ equals $s_2$. Thus $\mathbf{Trc}$ is a CPO, under the pointwise ordering: $T_1 \sqsubseteq T_2$ iff for all $s \in \mathbf{Sto}$, $T_1\, s \sqsubseteq T_2\, s$.

**Semantics.** For expressions, we assume there exists a semantic function $[\![E]\!] : \mathbf{Sto} \to \mathbf{Val}$ which satisfies the following property:

**Property 2.1** *If for all* $x \in \mathrm{fv}(E)$ *we have* $s_1\, x = s_2\, x$*, with* $s_1, s_2 \in \mathbf{Sto}$*, then* $[\![E]\!]s_1 = [\![E]\!]s_2$.

The definition of $[\![E]\!]$ would contain the clause $[\![x]\!]s = s\, x$. The semantics of a command has functionality $[\![C]\!] : \mathbf{Trc} \to \mathbf{Trc}$, and is defined in Fig. 2. Since $\mathbf{Trc}$ is a CPO also $\mathbf{Trc} \to \mathbf{Trc}$ is a CPO, with the following pointwise ordering: $f_1 \sqsubseteq f_2$ iff $f_1(T) \sqsubseteq f_2(T)$ for all $T \in \mathbf{Trc}$.

Two comments regarding Fig. 2: First, to streamline the treatment of $\bot$, the metalanguage expression "$\mathsf{let}\ \alpha = \beta\ \mathsf{in}\ \ldots$" denotes $\bot$ if $\beta$ is $\bot$.

$$
\begin{aligned}
[\![\mathsf{x} := \mathsf{E}]\!] &= \lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in}\ [\mathsf{s}' \mid \mathsf{x} \mapsto [\![\mathsf{E}]\!]\mathsf{s}'] \\
[\![\mathsf{C}_1\,;\mathsf{C}_2]\!] &= \lambda\mathsf{T}.[\![\mathsf{C}_2]\!]([\![\mathsf{C}_1]\!]\mathsf{T}) \\
[\![\textbf{if}\ \mathsf{E}\ \textbf{then}\ \mathsf{C}_1\ \textbf{else}\ \mathsf{C}_2]\!] &= \lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in} \\
&\qquad \mathsf{if}\ \mathit{true?}([\![\mathsf{E}]\!]\mathsf{s}')\ \mathsf{then}\ [\![\mathsf{C}_1]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{else}\ [\![\mathsf{C}_2]\!]\mathsf{T}\ \mathsf{s} \\
[\![\textbf{while}\ \mathsf{E}\ \textbf{do}\ \mathsf{C}_0]\!] &= \mathit{lfp}(\mathcal{F}) \quad \text{where} \\
&\qquad \mathcal{F} : (\mathbf{Trc} \to \mathbf{Trc}) \to (\mathbf{Trc} \to \mathbf{Trc}) \text{ is given by} \\
&\qquad \mathcal{F}(\mathsf{f}) = \lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in} \\
&\qquad\qquad \mathsf{if}\ \mathit{true?}([\![\mathsf{E}]\!]\mathsf{s}')\ \mathsf{then}\ \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}\ \mathsf{else}\ \mathsf{s}'
\end{aligned}
$$

Fig. 2. The Trace Semantics.

Second, in order to show that the least fixed point in the definition of **while** is well-defined, $\mathcal{F}$ (called the *functor* of the **while** command) ought to be continuous. But this follows from the following calculation:

$$
\begin{aligned}
\mathcal{F}(\sqcup_i \mathsf{f}_i) &= \lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in}\ (\mathsf{if}\ \mathit{true?}([\![\mathsf{E}]\!]\mathsf{s}')\ \mathsf{then}\ (\sqcup_i \mathsf{f}_i)([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}\ \mathsf{else}\ \mathsf{s}') \\
&= \lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in}\ (\mathsf{if}\ \mathit{true?}([\![\mathsf{E}]\!]\mathsf{s}')\ \mathsf{then}\ \sqcup_i (\mathsf{f}_i([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s})\ \mathsf{else}\ \mathsf{s}') \\
&= \sqcup_i(\lambda\mathsf{T}.\lambda\mathsf{s}.\mathsf{let}\ \mathsf{s}' = \mathsf{T}\ \mathsf{s}\ \mathsf{in}\ (\mathsf{if}\ \mathit{true?}([\![\mathsf{E}]\!]\mathsf{s}')\ \mathsf{then}\ \mathsf{f}_i([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}\ \mathsf{else}\ \mathsf{s}') \\
&= \sqcup_i(\mathsf{F}(\mathsf{f}_i))
\end{aligned}
$$

Therefore, with $\mathsf{C}$ of the form **while** $\mathsf{E}$ **do** $\mathsf{C}_0$,

$$[\![\mathsf{C}]\!] = \mathit{lfp}(\mathcal{F}) = \bigsqcup_{i \in \mathcal{N}} \mathsf{f}_i$$

where $\mathsf{f}_i$ (called an *iterand* of the **while** command) is defined by:

$$
\begin{aligned}
\mathsf{f}_0 &= \lambda\mathsf{T}.\lambda\mathsf{s}.\bot \\
\mathsf{f}_{i+1} &= \mathcal{F}(\mathsf{f}_i)
\end{aligned}
$$

We say that a function $\mathsf{f} \in \mathbf{Trc} \to \mathbf{Trc}$ is *fully strict* if for all $\mathsf{T} \in \mathbf{Trc}$ and $\mathsf{s} \in \mathbf{Sto}$, $\mathsf{T}\ \mathsf{s} = \bot$ implies $\mathsf{f}\ \mathsf{T}\ \mathsf{s} = \bot$.

We say that a fully strict function $\mathsf{f}$ *preserves* $\mathsf{z} \in \mathbf{Var}$ if whenever $\mathsf{f}\ \mathsf{T}\ \mathsf{s} \neq \bot$ (and thus $\mathsf{T}\ \mathsf{s} \neq \bot$) then $\mathsf{f}\ \mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$.

We now have the following results about the semantic functions:

**Fact 2.2** *For all commands* $\mathsf{C}$*, the function* $[\![\mathsf{C}]\!]$ *is monotone and fully strict. Also, all iterands* $\mathsf{f_i}$ *of* **while** *commands are monotone and fully strict.*

**PROOF.** We go by structural induction on $\mathsf{C}$. For **while**, we show that $\mathcal{F}$ maps monotone functions into monotone functions and fully strict functions into fully strict functions; the result then follows (as $\mathsf{f_0}$ is clearly monotone and fully strict) since the limit of monotone functions is itself monotone, and the limit of fully strict functions is itself fully strict.

**Lemma 2.3** *For all commands* $\mathsf{C}$*, and all* $\mathsf{z} \in \mathbf{Var}$ *with* $\mathsf{z} \notin \mathit{modified}(\mathsf{C})$*,*

- $[\![\mathsf{C}]\!]$ *preserves* $\mathsf{z}$*;*
- *if* $\mathsf{C}$ *is a* **while** *command then all its iterands* $\mathsf{f_i}$ *preserve* $\mathsf{z}$*.*

**PROOF.** Structural induction in $\mathsf{C}$, with a case analysis. In all cases we are given $\mathsf{T}$ and $\mathsf{s}$, and assume that $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s} \neq \bot$ (and by Fact 2.2 thus $\mathsf{T}\ \mathsf{s} \neq \bot$); we must show that $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$.

$\underline{\mathsf{C} = \mathsf{x} := \mathsf{E}.}$ From $\mathsf{z} \notin \mathit{modified}(\mathsf{C})$ we infer that $\mathsf{z} \neq \mathsf{x}$, and the claim is trivial.

$\underline{\mathsf{C} = \mathsf{C_1}\ ;\ \mathsf{C_2}.}$ Since $\mathsf{z} \notin \mathit{modified}(\mathsf{C_2})$, we infer inductively that $[\![\mathsf{C_2}]\!]([\![\mathsf{C_1}]\!]\mathsf{T})\ \mathsf{s}\ \mathsf{z} = [\![\mathsf{C_1}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z}$. Since $\mathsf{z} \notin \mathit{modified}(\mathsf{C_1})$, we infer inductively that $[\![\mathsf{C_1}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$. We thus get $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = [\![\mathsf{C_2}]\!]([\![\mathsf{C_1}]\!]\mathsf{T})\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$, as desired.

$\underline{\mathsf{C} = \textbf{if}\ \mathsf{E}\ \textbf{then}\ \mathsf{C_1}\ \textbf{else}\ \mathsf{C_2}.}$ Wlog. we can assume that $\mathit{true?}([\![\mathsf{E}]\!](\mathsf{T}\ \ \mathsf{s}))$, in which case $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = [\![\mathsf{C_1}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z}$. Since $\mathsf{z} \notin \mathit{modified}(\mathsf{C_1})$, we infer inductively that $[\![\mathsf{C_1}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$. This yields the claim.

$\underline{\mathsf{C} = \textbf{while}\ \mathsf{E}\ \textbf{do}\ \mathsf{C_0}.}$ Let $\mathsf{f_i}$ be the iterands of $\mathsf{C}$; our first task is to prove by induction in $\mathsf{i}$ that each $\mathsf{f_i}$ preserves $\mathsf{z}$. For $\mathsf{i} = 0$, the claim follows vacuously since $\mathsf{f_i}\ \mathsf{T}\ \mathsf{s} = \bot$. For the inductive case, we split into two cases, in both cases assuming $\mathsf{f_{i+1}}\ \mathsf{T}\ \mathsf{s} \neq \bot$ (and by Fact 2.2 thus $\mathsf{T}\ \mathsf{s} \neq \bot$):

- if $\mathit{true?}([\![\mathsf{E}]\!](\mathsf{T}\ \mathsf{s}))$ then

  $$\mathsf{f_{i+1}}\ \mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{f_i}([\![\mathsf{C_0}]\!]\mathsf{T})\ \mathsf{s}\ \mathsf{z} = [\![\mathsf{C_0}]\!]\mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$$

  where the second equality follows from the induction hypothesis on $\mathsf{f_i}$, and the third equality follows from the overall induction hypothesis on $\mathsf{C_0}$ (applicable since $\mathsf{z} \notin \mathit{modified}(\mathsf{C_0})$).
- if $\mathit{true?}([\![\mathsf{E}]\!](\mathsf{T}\ \mathsf{s}))$ does not hold, then $\mathsf{f_{i+1}}\ \mathsf{T}\ \mathsf{s}\ \mathsf{z} = \mathsf{T}\ \mathsf{s}\ \mathsf{z}$ follows directly.

We are left with showing that also $[\![C]\!]$, given as the least upper bound of the iterands, preserves $z$. But since $[\![C]\!]\mathsf{T}\ s \neq \bot$, we infer that there exists a natural number $j$ such that $[\![C]\!]\mathsf{T}\ s = \mathsf{f}_j\ \mathsf{T}\ s$, yielding the desired $[\![C]\!]\mathsf{T}\ s\ z = \mathsf{f}_j\ \mathsf{T}\ s\ z = \mathsf{T}\ s\ z$.

## 3 Independences

The trace semantics says that the meaning of a command is a trace transformer: it transforms initial (or pre) traces into current (or post) traces. In this section we will be interested in a finite abstraction of the pre-traces and the post-traces relevant to the execution of a command. The abstract traces are termed *independences*: an independence $\mathsf{T}^\# \in \mathbf{Independ} = \mathcal{P}((\mathbf{Var} \cup \{\bot\}) \times \mathbf{Var})$ is a set of pairs of the form $[x \ltimes w]$. If $x$ is a variable, then $[x \ltimes w]$ denotes that the *current* value of $x$ is independent of the *initial* value of $w$. If $x$ is $\bot$, then the *nontermination behavior* of the command is independent of $w$. This is formalized by the following definition which states the condition under which an independence correctly describes a concrete trace.

**Definition 3.1** *For all* $\mathsf{T} \in \mathbf{Trc}$*, for all* $x \in \mathsf{Var} \cup \{\bot\}$*, for all* $w \in \mathsf{Var}$*,* $\mathsf{T} \models [x \ltimes w]$ *holds iff for all* $s_1, s_2 \in \mathbf{Sto}$*:* $s_1 \overset{=}{w} s_2$ *implies* $\mathsf{T}\ s_1 \overset{x}{=} \mathsf{T}\ s_2$.

$\mathsf{T} \models \mathsf{T}^\#$ *holds iff for all* $[x \ltimes w] \in \mathsf{T}^\#$ *it holds that* $\mathsf{T} \models [x \ltimes w]$.

In the definition of $\mathsf{T} \models [x \ltimes w]$, note that $s_1, s_2$ are the "initial" stores and the antecedent of the implication asserts that these stores are equal except for $w$. The "current" stores are obtained via $\mathsf{T}\ s_1$ and $\mathsf{T}\ s_2$ and the consequent of the implication demands that the current stores be equal on $x$. Because $x$ can be $\bot$, and because of the way $[\bot \ltimes w]$ is defined, an analysis based on Definition 3.1 is *nontermination sensitive*. In particular, observe the following result:

**Fact 3.2** *Assume that* $\mathsf{T} \models \{[x \ltimes w], [\bot \ltimes w]\}$*. Then for all* $s_1, s_2 \in \mathbf{Sto}$*,* $s_1 \overset{=}{w} s_2$ *implies that either* $\mathsf{T}\ s_1 = \mathsf{T}\ s_2 = \bot$*, or* $\mathsf{T}\ s_1 \neq \bot$ *and* $\mathsf{T}\ s_2 \neq \bot$ *with* $\mathsf{T}\ s_1\ x = \mathsf{T}\ s_2\ x$.

**Definition 3.3** *The ordering* $\mathsf{T}_1^\# \preceq \mathsf{T}_2^\#$ *holds iff* $\mathsf{T}_2^\# \subseteq \mathsf{T}_1^\#$.

The intuition behind the definition is that in $\mathsf{T}_1^\# \preceq \mathsf{T}_2^\#$, $\mathsf{T}_1^\#$ is more precise than $\mathsf{T}_2^\#$ because $\mathsf{T}_1^\#$ deems more variables independent than $\mathsf{T}_2^\#$. A motivation for the definition is the desire for a subtyping rule, stating that if $\mathsf{T}_1^\# \preceq \mathsf{T}_2^\#$ then $\mathsf{T}_1^\#$ can be replaced by $\mathsf{T}_2^\#$ (c.f. Fact 3.4). The subtyping rule is sound provided $\mathsf{T}_2^\#$ is a subset of $\mathsf{T}_1^\#$ and therefore obtainable from $\mathsf{T}_1^\#$ by removing information.

Clearly, **Independ** forms a complete lattice wrt. the ordering $\preceq$; let $\sqcap_i \mathsf{T}^{\#}{}_i$ denote the greatest lower bound (which is the set union). The greatest element is the empty set of independences and the least element is **Independ**; the least element is obtained when all variables have constant values. We have some expected properties, where Fact 3.4 is used in the proof of the correctness theorem (Section 5), and where Fact 3.5 follows since if $[\mathsf{x} \ltimes \mathsf{w}]$ belongs to $\sqcap_i \mathsf{T}_i^{\#}$ then it also belongs to some $\mathsf{T}_i^{\#}$.

**Fact 3.4** *If* $\mathsf{T} \models \mathsf{T}_1^{\#}$ *and* $\mathsf{T}_1^{\#} \preceq \mathsf{T}_2^{\#}$ *then* $\mathsf{T} \models \mathsf{T}_2^{\#}$.

**Fact 3.5** *If for all* $\mathsf{i} \in \mathsf{I}$ *it holds that* $\mathsf{T} \models \mathsf{T}_i^{\#}$, *then* $\mathsf{T} \models \sqcap_{i \in I} \mathsf{T}_i^{\#}$.

We have the following fact about the identity trace.

**Fact 3.6** *For all* $\mathsf{x} \in \mathbf{Var} \cup \{\bot\}$, *for all* $\mathsf{y} \in \mathbf{Var}$, *if* $\mathsf{x} \neq \mathsf{y}$ *then* $\lambda \mathsf{s}.\mathsf{s} \models [\mathsf{x} \ltimes \mathsf{y}]$.

**Do we have an abstract interpretation?** Facts 3.4 and 3.5 lead us to explore the connection of our framework with abstract interpretation [10]. We can write a function $\gamma : \mathbf{Independ} \to \mathcal{P}(\mathbf{Trc})$:

$$\gamma(\mathsf{T}^{\#}) = \{\mathsf{T} \in \mathbf{Trc} \mid \mathsf{T} \models \mathsf{T}^{\#}\}$$

and demonstrate that $\gamma$ is completely multiplicative. We calculate:

$$\begin{aligned}
\mathsf{T} \in \gamma(\sqcap_i \mathsf{T}^{\#}{}_i) &\Leftrightarrow \forall [\mathsf{x} \ltimes \mathsf{y}] \in \sqcap_i \mathsf{T}^{\#}{}_i \bullet \mathsf{T} \models [\mathsf{x} \ltimes \mathsf{y}] \\
&\Leftrightarrow \forall \mathsf{i} \bullet \forall [\mathsf{x} \ltimes \mathsf{y}] \in \mathsf{T}_i^{\#} \bullet \mathsf{T} \models [\mathsf{x} \ltimes \mathsf{y}] \\
&\Leftrightarrow \forall \mathsf{i} \bullet \mathsf{T} \in \gamma(\mathsf{T}_i^{\#}) \\
&\Leftrightarrow \mathsf{T} \in \bigcap_i \gamma(\mathsf{T}_i^{\#})
\end{aligned}$$

Therefore, $\gamma$ can be considered a concretization function so that, by properties of Galois Connections, e.g., [21, Lemma 4.23], there exists an abstraction function $\alpha : \mathcal{P}(\mathbf{Trc}) \to \mathbf{Independ}$:

$$\alpha(\mathsf{U}) = \bigcup \{\mathsf{T}^{\#} \mid \mathsf{U} \subseteq \gamma(\mathsf{T}^{\#})\}$$

such that $(\mathcal{P}(\mathbf{Trc}), \alpha, \gamma, \mathbf{Independ})$ is a Galois connection.

Equivalently, Facts 3.4 and 3.5 state that relation $\models$ is $\mathsf{U}$-closed (upwards closed) and $\mathsf{G}$-closed (greatest lower bound closed) respectively. Therefore, $\models$ defines a Galois connection [30]. It follows that by defining $\gamma(\mathsf{T}^{\#})$ as above, we have the properties:

- $\gamma$ is a monotone function.
- For all $\mathsf{T}^{\#} \in \mathbf{Independ}$, for all $\mathsf{U} \in \mathcal{P}(\mathbf{Trc})$, $\mathsf{U} \subseteq \gamma(\mathsf{T}^{\#})$ iff $\alpha(\mathsf{U}) \preceq \mathsf{T}^{\#}$.

The second item above is the definition of a Galois connection and in this sense, $\models$ "is" a Galois connection.

## 4 Static Checking of Independences

To statically check independences we define, in Fig. 3, a Hoare-like logic where judgements are of the form

$$\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$$

The judgement is interpreted as saying that if the independences in $\mathsf{T}_0^{\#}$ hold *before* execution of $\mathsf{C}$ then the independences in $\mathsf{T}^{\#}$ will hold *after* execution of $\mathsf{C}$. The context $\mathsf{G} \in \mathbf{Context} = \mathcal{P}(\mathbf{Var})$ is a *control dependence*, denoting (a superset of) the variables that at least one test surrounding $\mathsf{C}$ depends on. For example, in **if** $\mathsf{x}$ **then** $\mathsf{y} := 0$ **else** $\mathsf{z} := 1$, the static checking of $\mathsf{y} := 0$ takes place in the context that contains all variables that $\mathsf{x}$ is dependent on. This is crucial, especially since $\mathsf{x}$ may depend on a high variable.

We now explain a few of the rules in Fig. 3. Checking an assignment, $\mathsf{x} := \mathsf{E}$, in context $\mathsf{G}$, involves checking any $[\mathsf{y} \ltimes w]$ in the postcondition $\mathsf{T}^{\#}$. There are two cases. If $\mathsf{x} \neq \mathsf{y}$, then $[\mathsf{y} \ltimes w]$ must also appear in the precondition $\mathsf{T}_0^{\#}$. Likewise, if $[\bot \ltimes w]$ appears in the postcondition $\mathsf{T}^{\#}$, then it must also appear in the precondition $\mathsf{T}_0^{\#}$. Otherwise, if $\mathsf{x} = \mathsf{y}$ then $[\mathsf{x} \ltimes w]$ appears in the postcondition provided all variables referenced in $\mathsf{E}$ are independent of $w$; moreover, $w$ must not appear in $\mathsf{G}$, as otherwise, $\mathsf{x}$ would be control dependent on $w$.

Checking a conditional, **if** $\mathsf{E}$ **then** $\mathsf{C}_1$ **else** $\mathsf{C}_2$, involves checking $\mathsf{C}_1$ and $\mathsf{C}_2$ in a context $\mathsf{G}_0$ that includes not only the "old" context $\mathsf{G}$ but also the variables that $\mathsf{E}$ depends on (as variables modified in $\mathsf{C}_1$ or $\mathsf{C}_2$ will be control dependent on the variables that $\mathsf{E}$ depends on.) Equivalently, if $w$ is not in $\mathsf{G}_0$, then all free variables $\mathsf{x}$ in $\mathsf{E}$ must be independent of $w$, that is, $[\mathsf{x} \ltimes w]$ must appear in the precondition $\mathsf{T}_0^{\#}$.

Checking a while loop is similar to checking a conditional. The only difference is that it requires guessing an "invariant" $\mathsf{T}^{\#}$ that is both the precondition and the postcondition of the loop and its body. With respect to nontermination, note that if $w \in \mathsf{G}_0$ then $w$ may influence the termination of the program: either directly, as in **while** $w > 7$ **do** $w := w + 1$ (where the side condition for [While] forces $w$ to be added to $\mathsf{G}_0$); or indirectly, as in

$$[\mathsf{Assign}]\quad \mathsf{G} \vdash \{\mathsf{T}_0^\#\}\, \mathsf{x} := \mathsf{E}\, \{\mathsf{T}^\#\} \qquad \begin{array}{l}\text{if } \forall [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\# \bullet \\ \quad (\mathsf{x} \neq \mathsf{y} \vee \mathsf{y} = \bot) \Rightarrow [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\# \\ \quad \mathsf{x} = \mathsf{y} \Rightarrow (\mathsf{w} \notin \mathsf{G} \wedge \forall \mathsf{z} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{z} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#)\end{array}$$

$$[\mathsf{Seq}]\quad \frac{\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\, \mathsf{C}_1\, \{\mathsf{T}_1^\#\} \quad \mathsf{G} \vdash \{\mathsf{T}_1^\#\}\, \mathsf{C}_2\, \{\mathsf{T}_2^\#\}}{\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\, \mathsf{C}_1 ; \mathsf{C}_2\, \{\mathsf{T}_2^\#\}}$$

$$[\mathsf{If}]\quad \frac{\mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\, \mathsf{C}_1\, \{\mathsf{T}^\#\} \quad \mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\, \mathsf{C}_2\, \{\mathsf{T}^\#\}}{\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\, \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathsf{C}_1\ \mathbf{else}\ \mathsf{C}_2\, \{\mathsf{T}^\#\}} \quad \text{if} \quad \begin{array}{l}\mathsf{G} \subseteq \mathsf{G}_0 \wedge \\ (\mathsf{w} \notin \mathsf{G}_0 \Rightarrow \\ \forall \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#)\end{array}$$

$$[\mathsf{While}]\quad \frac{\mathsf{G}_0 \vdash \{\mathsf{T}^\#\}\, \mathsf{C}\, \{\mathsf{T}^\#\}}{\mathsf{G} \vdash \{\mathsf{T}^\#\}\, \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathsf{C}\, \{\mathsf{T}^\#\}} \quad \text{if} \quad \begin{array}{l}\mathsf{G} \subseteq \mathsf{G}_0 \wedge \\ (\mathsf{w} \notin \mathsf{G}_0 \Rightarrow \\ \forall \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \in \mathsf{T}^\#) \wedge \\ ([\bot \ltimes \mathsf{w}] \in \mathsf{T}^\# \Rightarrow \mathsf{w} \notin \mathsf{G}_0)\end{array}$$

$$[\mathsf{Sub}]\quad \frac{\mathsf{G}_1 \vdash \{\mathsf{T}_1^\#\}\, \mathsf{C}\, \{\mathsf{T}_2^\#\}}{\mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\, \mathsf{C}\, \{\mathsf{T}_3^\#\}} \quad \text{if } (\mathsf{T}_0^\# \preceq \mathsf{T}_1^\#) \wedge (\mathsf{T}_2^\# \preceq \mathsf{T}_3^\#) \wedge (\mathsf{G}_0 \subseteq \mathsf{G}_1)$$

Fig. 3. The Logic.

**if** $w > 7$ **then skip else while** *true* **do skip** (where the side condition for [If] has already added $w$ to G). Therefore we demand that if $[\bot \ltimes w] \in \mathsf{T}^\#$ then $w$ must not be in $\mathsf{G}_0$.

In Section 7, when we define *strongest postcondition*, we will select $\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \notin \mathsf{T}_0^\#\}$ for the conditional and the while loop. Instead of guessing the invariant, we will show how to compute it using fixpoints.

**Example 4.1** *(Illustrating "recovery" of independences.)* We have the derivations

$$\emptyset \vdash \{\{[\mathtt{l} \ltimes \mathtt{h}], [\mathtt{h} \ltimes \mathtt{l}]\}\}\, \mathtt{l} := \mathtt{h}\, \{\{[\mathtt{h} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{l}]\}\} \text{ and}$$
$$\emptyset \vdash \{\{[\mathtt{h} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{l}]\}\}\, \mathtt{l} := 0\, \{\{[\mathtt{h} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{h}]\}\}$$

and therefore also

$$\emptyset \vdash \{\{[\mathtt{l} \ltimes \mathtt{h}], [\mathtt{h} \ltimes \mathtt{l}]\}\}\, \mathtt{l} := \mathtt{h} ; \mathtt{l} := 0\, \{\{[\mathtt{h} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{l}], [\mathtt{l} \ltimes \mathtt{h}]\}\}.$$

With the intuition that $\mathtt{l}$ stands for "low" or "public" and $\mathtt{h}$ stands for "high" or "sensitive", the derivation asserts that if $\mathtt{l}$ is independent of $\mathtt{h}$ before execution, then $\mathtt{l}$ is independent of $\mathtt{h}$ after execution. Thus $[\mathtt{l} \ltimes \mathtt{h}]$ is an *invariant*

of the computation. By Definition 3.1, any trace of the program applied to initial stores that differ only in the value for $\mathtt{h}$, creates new stores that agree on the current value for $\mathtt{l}$. Thus the program is secure, although it contains an insecure sub-program: the independence $[\mathtt{l} \ltimes \mathtt{h}]$ is lost after $\mathtt{l} := \mathtt{h}$, but recovered after $\mathtt{l} := 0$.

**Example 4.2** *(Illustrating control dependence.)* The reader may check that the following informally annotated program gives rise to a derivation in our logic. Initially, $\mathsf{G}$ is empty, and all variables are pairwise independent; we write $[\mathtt{x} \ltimes \mathtt{y}, \mathtt{z}]$ to abbreviate $[\mathtt{x} \ltimes \mathtt{y}], [\mathtt{x} \ltimes \mathtt{z}]$.

$$
\begin{array}{ll}
 & \{[\mathtt{l} \ltimes \mathtt{h}, \mathtt{x}], [\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}], [\mathtt{x} \ltimes \mathtt{l}, \mathtt{h}]\} \\
\mathtt{x} := \mathtt{h} & \{[\mathtt{l} \ltimes \mathtt{h}, \mathtt{x}], [\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}], [\mathtt{x} \ltimes \mathtt{l}, \mathtt{x}]\} \\
\mathtt{if}\ \mathtt{x} > 0 & (\mathsf{G} \text{ is now } \{\mathtt{h}\}) \\
\quad \mathtt{then}\ \mathtt{l} := 7 & \{[\mathtt{l} \ltimes \mathtt{x}, \mathtt{l}], [\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}], [\mathtt{x} \ltimes \mathtt{l}, \mathtt{x}]\} \\
\quad \mathtt{else}\ \mathtt{x} := 0 & \{[\mathtt{l} \ltimes \mathtt{h}, \mathtt{x}], [\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}], [\mathtt{x} \ltimes \mathtt{l}, \mathtt{x}]\} \\
\textit{end of}\ \mathtt{if} & \{[\mathtt{l} \ltimes \mathtt{x}], [\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}], [\mathtt{x} \ltimes \mathtt{l}, \mathtt{x}]\}
\end{array}
$$

A few remarks:

- in the preamble, only $\mathtt{x}$ is assigned, so the independences for $\mathtt{l}$ and $\mathtt{h}$ are carried through, but $[\mathtt{x} \ltimes \mathtt{l}, \mathtt{x}]$ holds afterwards, as $[\mathtt{h} \ltimes \mathtt{l}, \mathtt{x}]$ holds beforehand;
- the free variable in the guard is independent of $\mathtt{l}$ and $\mathtt{x}$ but not of $\mathtt{h}$, implying that $\mathtt{h}$ has to be in $\mathsf{G}$.

**Example 4.3** *(Illustrating how the logic works with nontermination.)*

With $\mathsf{P} = \mathbf{while}\ \mathtt{l} \neq 0\ \mathbf{do}\ \mathtt{h} := 7$ and $\mathsf{T}^{\#} = \{[\bot \ltimes \mathtt{h}], [\mathtt{l} \ltimes \mathtt{h}]\}$ we have the judgement $\emptyset \vdash \{\mathsf{T}^{\#}\}\ \mathsf{P}\ \{\mathsf{T}^{\#}\}$ (since $\{\mathtt{l}\} \vdash \{\mathsf{T}^{\#}\}\ \mathtt{h} := 7\ \{\mathsf{T}^{\#}\}$) showing that $\mathsf{P}$ is deemed secure by our logic; an observer able to observe even nontermination cannot detect the initial value of $\mathtt{h}$. (The reason why $[\mathtt{h} \ltimes \mathtt{l}]$ and $[\bot \ltimes \mathtt{l}]$ are not in $\mathsf{T}^{\#}$ is that $\mathtt{h}$ clearly depends on $\mathtt{l}$ and nontermination depends on $\mathtt{l}$.)

However, with $\mathsf{P} = \mathbf{while}\ \mathtt{h} \neq 0\ \mathbf{do}\ \mathtt{h} := 7$ and $\mathsf{T}^{\#} = \{[\bot \ltimes \mathtt{h}], [\mathtt{l} \ltimes \mathtt{h}]\}, [\mathtt{h} \ltimes \mathtt{l}]$ we do not have a derivation, even though $\mathsf{T}^{\#}$ is an invariant for $\mathtt{h} := 7$. But because the derivation requires $\mathtt{h} \in \mathsf{G}_0$, $[\bot \ltimes \mathtt{h}]$ can no longer be in $\mathsf{T}^{\#}$. This suggests that $\mathsf{P}$ is insecure when nontermination is observable: indeed, an observer able to observe nontermination can detect whether $\mathtt{h}$ was initially 0 or not. Interestingly, if nontermination is *not* observable, then $\mathsf{T}^{\#} = \{[\mathtt{l} \ltimes \mathtt{h}], [\mathtt{h} \ltimes \mathtt{l}]\}$ remains invariant, as expected. In particular, the current value of $\mathtt{l}$ is independent of the initial value of $\mathtt{h}$. Similarly, we do not

have a derivation for $\mathsf{P} = \textbf{if } \mathsf{h} \neq 0 \textbf{ then skip else while } \mathit{true} \textbf{ do skip}$ with $\mathsf{T}^{\#}$ containing $[\bot \ltimes \mathsf{h}]$.

A couple of simple results can be proven about the logic in Fig. 3:

**Fact 4.4** *Assume* $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$. *If* $[\bot \ltimes \mathsf{w}] \in \mathsf{T}^{\#}$ *then* $[\bot \ltimes \mathsf{w}] \in \mathsf{T}_0^{\#}$.

The proof is an easy induction on a derivation of $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$.

**Lemma 4.5** *Assume* $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$ *with* $[\bot \ltimes \mathsf{w}] \in \mathsf{T}^{\#}$ *and* $\mathsf{w} \in \mathsf{G}$. *Then* $\mathsf{T}\ \mathsf{s} \neq \bot$ *implies* $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s} \neq \bot$.

**PROOF.** An easy induction in the derivation, where for [Seq] we use Fact 4.4; for [While] the result follows vacuously since having both $[\bot \ltimes \mathsf{w}] \in \mathsf{T}^{\#}$ and $\mathsf{w} \in \mathsf{G}$ violates the side conditions of that rule.

To restate Fact 4.4, if $[\bot \ltimes \mathsf{w}] \notin \mathsf{T}_0^{\#}$, then $[\bot \ltimes \mathsf{w}]$ cannot be recovered in $\mathsf{T}^{\#}$. This is in contrast to Example 4.1, which shows that although $[\mathsf{l} \ltimes \mathsf{h}]$ does not appear in the precondition of $\mathsf{l} := 0$, it could nonetheless be recovered in the postcondition of $\mathsf{l} := 0$. In other words, once it is established that nontermination may depend on $\mathsf{w}$, it continues to depend on $\mathsf{w}$. This is what we should expect, as otherwise, with $\mathsf{w}$ interpreted "high", nontermination can be used to leak the high value.

In a judgement $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$, suppose $\mathsf{w} \in \mathsf{G}$. This means that any assignment in $\mathsf{C}$ is control dependent on $\mathsf{w}$. Suppose now that $\mathsf{y}$ is a variable and is independent of $\mathsf{w}$ in the postcondition $\mathsf{T}^{\#}$. This implies that $\mathsf{y}$ cannot be assigned to in $\mathsf{C}$ — otherwise, it would be dependent on $\mathsf{w}$. If $\mathsf{y}$ is not assigned to in $\mathsf{C}$, then $\mathsf{y}$ must be independent of $\mathsf{w}$ in the precondition too. These intuitions are collected together in Lemma 4.6 below. Note that with $\mathsf{y}$ interpreted as "low" and $\mathsf{w}$ as "high", the lemma essentially says that low variables may not be written to under a high guard. Thus the lemma is the counterpart of the "no write down" rule that underlies information flow control; the term "*-property" [7] is also used. The value of low variables remains the same after execution of $\mathsf{C}$.

**Lemma 4.6 (Write Confinement)** *Assume that* $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$. *Then the following holds:*

> *If* $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^{\#}$ *and* $\mathsf{y} \in \textbf{Var}$ *and* $\mathsf{w} \in \mathsf{G}$
> *then* $\mathsf{y} \notin \mathit{modified}(\mathsf{C})$ *and* $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^{\#}$.

**PROOF.** We perform induction in the derivation of $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$, and do a case analysis on the last rule applied:

[Assign], with $\mathsf{C} = \mathsf{x} := \mathsf{E}$. If $\mathsf{x} = \mathsf{y}$, then $\mathsf{w} \notin \mathsf{G}$, contradicting our assumptions. If $\mathsf{x} \neq \mathsf{y}$, then $\mathsf{y} \notin \mathit{modified}(\mathsf{C})$ and $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$.

[Seq], with $\mathsf{C} = \mathsf{C}_1\,;\mathsf{C}_2$. Assume that $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}_1^\#\}$ and that $\mathsf{G} \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}_2\ \{\mathsf{T}^\#\}$ and also assume that $\mathsf{w} \in \mathsf{G}$. By applying the induction hypothesis to the latter judgement, we see that $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_2)$ and that $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_1^\#$. By then applying the induction hypothesis to the former judgement, we see that $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_1)$ and that $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$. Therefore $\mathsf{y} \notin \mathit{modified}(\mathsf{C})$, as desired.

[If], with $\mathsf{C} = \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathsf{C}_1\ \mathbf{else}\ \mathsf{C}_2$. Assume that

$$\mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}^\#\} \text{ and } \mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_2\ \{\mathsf{T}^\#\}$$

where $\mathsf{G} \subseteq \mathsf{G}_0$. Let $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#$ with $\mathsf{w} \in \mathsf{G}$, then $\mathsf{w} \in \mathsf{G}_0$ so by applying the induction hypothesis we get $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_1)$ and $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_2)$ and $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$. This implies $\mathsf{y} \notin \mathit{modified}(\mathsf{C})$, and thereby the desired result.

[While], with $\mathsf{C} = \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathsf{C}_0$. Our assumptions are that $\mathsf{G} \vdash \{\mathsf{T}^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$ because with $\mathsf{G} \subseteq \mathsf{G}_0$ we have $\mathsf{G}_0 \vdash \{\mathsf{T}^\#\}\ \mathsf{C}_0\ \{\mathsf{T}^\#\}$. Let $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#$ with $\mathsf{w} \in \mathsf{G}$, then $\mathsf{w} \in \mathsf{G}_0$ so by applying the induction hypothesis we get $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_0)$ (and vacuously $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#$) which is as desired.

[Sub]. Assume that $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$ because with $\mathsf{G} \subseteq \mathsf{G}_0$ and $\mathsf{T}_0^\# \preceq \mathsf{T}_1^\#$ and $\mathsf{T}_2^\# \preceq \mathsf{T}^\#$ we have $\mathsf{G}_0 \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}\ \{\mathsf{T}_2^\#\}$. Also assume that $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#$ and that $\mathsf{w} \in \mathsf{G}$. Then $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_2^\#$ and $\mathsf{w} \in \mathsf{G}_0$, so inductively we obtain $\mathsf{y} \notin \mathit{modified}(\mathsf{C})$ and $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_1^\#$. Then also $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$, as desired, because $\mathsf{T}_0^\# \preceq \mathsf{T}_1^\#$.

## 5 Correctness

We are now in a position to prove the correctness of the logic with respect to the trace semantics.

**Theorem 5.1** *Assume that*

$\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$ *where for all* $[\mathsf{x} \ltimes \mathsf{y}] \in \mathsf{T}_0^\#$ *it is the case that* $\mathsf{x} \neq \mathsf{y}$. *Then* $[\![\mathsf{C}]\!](\lambda \mathsf{s}.\mathsf{s}) \models \mathsf{T}^\#$.

That is, $\mathsf{T}^{\#}$ correctly describes the concrete trace obtained by executing command $\mathsf{C}$ on $\lambda s.s$, the initial trace.

The correctness theorem can be seen as the noninterference theorem for information flow. Indeed, consider the trace, $\mathsf{T}$, obtained from executing the command $\mathsf{C}$ with the initial trace, $\lambda s.s$. With $\mathsf{l}$ and $\mathsf{h}$ interpreted as "low" and "high" respectively, suppose $[\mathsf{l} \ltimes \mathsf{h}]$ appears in $\mathsf{T}^{\#}$. Then for any two stores $s_1, s_2$ that differ only on $\mathsf{h}$, the stores $\mathsf{T}\ s_1$ and $\mathsf{T}\ s_2$ must agree on the value of $\mathsf{l}$, meaning that if both $\mathsf{T}\ s_1$ and $\mathsf{T}\ s_2$ terminate then they must not give different values for $\mathsf{l}$. On the other hand, it is possible for either $\mathsf{T}\ s_1$ or $\mathsf{T}\ s_2$ or both to be $\perp$, as in the example

**if** $\mathsf{h}$ **then** $\mathsf{h} := 7$ **else** (**while** *true* **do skip**).

which has $[\mathsf{l} \ltimes \mathsf{h}]$ as an invariant.

Moreover, the correctness theorem says that if $[\perp \ltimes \mathsf{h}]$ appears in $\mathsf{T}^{\#}$, then $\mathsf{T}\ s_1 = \perp$ iff $\mathsf{T}\ s_2 = \perp$: thus noninterference is preserved under nontermination.

To prove Theorem 5.1, we need the following main lemma. Then the theorem follows by substituting $\mathsf{T}$ by $\lambda s.s$, and then using Fact 3.6.

**Lemma 5.2** *If* $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$ *and* $\mathsf{T} \models \mathsf{T}_0^{\#}$*, then* $[\![\mathsf{C}]\!]\mathsf{T} \models \mathsf{T}^{\#}$.

**PROOF.** We perform induction on a derivation of $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$, and do a case analysis on the last rule applied.

[Assign], with $\mathsf{C} = x := \mathsf{E}$. We are given $[z \ltimes w] \in \mathsf{T}^{\#}$, and we consider $s_1, s_2 \in$ **Sto** such that $s_1 \underset{w}{=} s_2$. With $\mathsf{T}' = [\![\mathsf{C}]\!]\mathsf{T}$, we must show that $\mathsf{T}'\ s_1 \overset{z}{=} \mathsf{T}'\ s_2$.

If $z = \perp$, then $[z \ltimes w] \in \mathsf{T}_0^{\#}$, so $\mathsf{T}\ s_1 \overset{\perp}{=} \mathsf{T}\ s_2$. Since for all $s$, $\mathsf{T}'\ s = \perp$ iff $\mathsf{T}\ s = \perp$, this shows that also $\mathsf{T}'\ s_1 \overset{\perp}{=} \mathsf{T}'\ s_2$, as desired.

So now consider $z \in$ **Var**. If either $\mathsf{T}\ s_1 = \perp$ or $\mathsf{T}\ s_2 = \perp$, also $\mathsf{T}'\ s_1 = \perp$ or $\mathsf{T}'\ s_2 = \perp$, and the claim is trivial. We thus assume that $\mathsf{T}\ s_1 \neq \perp$ and $\mathsf{T}\ s_2 \neq \perp$. We have two subcases:

(1) $z \neq x$: Then $[z \ltimes w] \in \mathsf{T}_0^{\#}$, so $\mathsf{T}\ s_1 \overset{z}{=} \mathsf{T}\ s_2$, that is, $\mathsf{T}\ s_1\ z = \mathsf{T}\ s_2\ z$. Since for all $s$ we have $\mathsf{T}'\ s\ z = \mathsf{T}\ s\ z$, this shows that also $\mathsf{T}'\ s_1\ z = \mathsf{T}'\ s_2\ z$, as desired.
(2) $z = x$: Then for all $y \in \mathrm{fv}(\mathrm{E})$, we have $[y \ltimes w] \in \mathsf{T}_0^{\#}$ and therefore $\mathsf{T}\ s_1\ y = \mathsf{T}\ s_2\ y$. By Property 2.1 then $[\![\mathsf{E}]\!](\mathsf{T}\ s_1) = [\![\mathsf{E}]\!](\mathsf{T}\ s_2)$, showing that $\mathsf{T}'\ s_1\ x = \mathsf{T}'\ s_2\ x$, as desired.

[Seq], with $C = C_1 ; C_2$. Assume that

$$G \vdash \{T_0^\#\}\, C_1\, \{T_1^\#\} \text{ and that } G \vdash \{T_1^\#\}\, C_2\, \{T_2^\#\}.$$

By applying the induction hypothesis to the first judgement, we get

$$[\![C_1]\!](T) \models T_1^\#.$$

We then apply the induction hypothesis to the second judgement and get the desired result:

$$[\![C_2]\!]([\![C_1]\!](T)) \models T^\#.$$

[If], with $C = \textbf{if } E \textbf{ then } C_1 \textbf{ else } C_2$. Assume that

$$G_0 \vdash \{T_0^\#\}\, C_1\, \{T^\#\} \text{ and } G_0 \vdash \{T_0^\#\}\, C_2\, \{T^\#\}$$

where $w \notin G_0$ implies that $\forall x \in \mathrm{fv}(E) \bullet [x \ltimes w] \in T_0^\#$. Inductively, we can assume that $[\![C_1]\!]T \models T^\#$ and $[\![C_2]\!]T \models T^\#$. We are given $[z \ltimes w] \in T^\#$ (where $z$ might be $\bot$), and we consider $s_1, s_2 \in \mathbf{Sto}$ such that $s_1 \underset{w}{=} s_2$. With $T' = [\![C]\!]T$, we must show that $T'\, s_1 \overset{z}{=} T'\, s_2$.

First assume that at least one of $T\, s_1$ or $T\, s_2$ is $\bot$. By Fact 2.2, at least as many of $T'\, s_1$ and $T'\, s_2$ will be $\bot$. So if $z \in \mathbf{Var}$, then trivially $T'\, s_1 \overset{z}{=} T'\, s_2$. If $z = \bot$, then by Fact 4.4 we know that $[\bot \ltimes w] \in T_0^\#$, so from $T \models T_0^\#$ we get $T\, s_1 \overset{\bot}{=} T\, s_2$ and therefore $T\, s_1 = T\, s_2 = \bot$. But then also $T'\, s_1 = T'\, s_2 = \bot$, showing $T'\, s_1 \overset{\bot}{=} T'\, s_2$.

We can thus assume that there exists stores $s_1', s_2' \neq \bot$ such that $s_1' = T\, s_1$ and $s_2' = T\, s_2$. Apart from symmetry, there are two cases:

***true?***$([\![E]\!]s_1')$ **and** ***true?***$([\![E]\!]s_2')$**.** Then $T'\, s_1 = [\![C_1]\!]T\, s_1$ and $T'\, s_2 = [\![C_1]\!]T\, s_2$. From $[\![C_1]\!]T \models T^\#$ and $[z \ltimes w] \in T^\#$ we infer that $[\![C_1]\!]T\, s_1 \overset{z}{=} [\![C_1]\!]T\, s_2$, which amounts to the desired $T'\, s_1 \overset{z}{=} T'\, s_2$.

***true?***$([\![E]\!]s_1')$ **but not** ***true?***$([\![E]\!]s_2')$**.** We claim

$$w \in G_0$$

and prove the claim by contradiction: suppose $w \notin G_0$. Then by the logic, for all $x \in \mathrm{fv}(E)$, $[x \ltimes w] \in T_0^\#$ implying $T\, s_1 \overset{x}{=} T\, s_2$, that is $s_1'(x) = s_2'(x)$. By Property 2.1 this implies $[\![E]\!]s_1' = [\![E]\!]s_2'$ – a contradiction.

Having established $w \in G_0$, consider two cases. First assume $z = \bot$. Then (by Lemma 4.5 applied to the sub-derivations) we infer that $[\![C_1]\!]T\, s_1 \neq \bot$ and

$[\![C_2]\!]T\ s_2 \neq \bot$, and thus $T'\ s_1 \neq \bot$ and $T'\ s_2 \neq \bot$. But this yields the desired relation $T'\ s_1 \stackrel{\bot}{=} T'\ s_2$.

Now assume that $z \in \mathbf{Var}$. By Lemma 4.6 applied to the sub-derivations, we infer that

$$[z \ltimes w] \in T_0^\# \text{ and } z \notin \mathit{modified}(C_1) \text{ and } z \notin \mathit{modified}(C_2).$$

Since $T \models T_0^\#$, this shows $T\ s_1 \stackrel{z}{=} T\ s_2$, that is, $T\ s_1\ z = T\ s_2\ z$. We know that $T'\ s_1 = [\![C_1]\!]T\ s_1$ and $T'\ s_2 = [\![C_2]\!]T\ s_2$. If either is $\bot$, the claim vacuously holds; so assume neither is $\bot$. By Lemma 2.3, $T\ s_1\ z = [\![C_1]\!]T\ s_1\ z$ and $T\ s_2\ z = [\![C_2]\!]T\ s_2\ z$. This shows the desired $T'\ s_1\ z = T\ s_1\ z = T\ s_2\ z = T'\ s_2\ z$.

[While], with $C = \mathbf{while}\ E\ \mathbf{do}\ C_0$. Our assumptions are that

$$G \vdash \{T^\#\}\ C\ \{T^\#\}$$

because with $G_0$ such that $w \notin G_0$ implies that $\forall x \in \mathrm{fv}(E) \bullet [x \ltimes w] \in T^\#$, and such that $[\bot \ltimes w] \in T^\#$ implies $w \notin G_0$, we have

$$G_0 \vdash \{T^\#\}\ C_0\ \{T^\#\}.$$

We define an auxiliary predicate $\mathcal{P}$:

$$\mathcal{P}(f) \Leftrightarrow \forall T \bullet (T \models T^\# \Rightarrow f\ T \models T^\#)$$

We shall establish

$$\forall i \geq 0 \bullet \mathcal{P}(f_i) \tag{1}$$

and do so by induction in $i$. For the base case, note that $f_0(T) = \lambda s.\bot$, and that $\lambda s.\bot \models T^\#$ always holds because no matter whether $z = \bot$ or not, $\bot \stackrel{z}{=} \bot$.

For the inductive case, we assume that $T \models T^\#$ and must prove $\mathcal{F}\ f\ T \models T^\#$, with $f$ an iterand. So let $[z \ltimes w] \in T^\#$ and $s_1 \underset{w}{=} s_2$; we know that $T\ s_1 \stackrel{z}{=} T\ s_2$ and must prove $\mathcal{F}\ f\ T\ s_1 \stackrel{z}{=} \mathcal{F}\ f\ T\ s_2$.

First assume that at least one of $T\ s_1$ or $T\ s_2$ is $\bot$. By Fact 2.2, at least as many of $\mathcal{F}\ f\ T\ s_1$ and $\mathcal{F}\ f\ T\ s_2$ will be $\bot$. So if $z \in \mathbf{Var}$, then trivially $\mathcal{F}\ f\ T\ s_1 \stackrel{z}{=} \mathcal{F}\ f\ T\ s_2$. If $z = \bot$, then from $[\bot \ltimes w] \in T^\#$ and $T \models T^\#$ we get $T\ s_1 \stackrel{\bot}{=} T\ s_2$ and therefore $T\ s_1 = T\ s_2 = \bot$. But then also $\mathcal{F}\ f\ T\ s_1 = \mathcal{F}\ f\ t\ s_2 = \bot$, showing $\mathcal{F}\ f\ T\ s_1 \stackrel{\bot}{=} \mathcal{F}\ f\ T\ s_2$.

We can thus assume that there exists stores $s_1', s_2' \neq \bot$ such that $s_1' = T\ s_1$ and $s_2' = T\ s_2$. Apart from symmetry, there are three cases:

***true?***$([\![\mathsf{E}]\!]\mathsf{s}_1')$ **and** ***true?***$([\![\mathsf{E}]\!]\mathsf{s}_2')$**.** Then $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1 = \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_1$ and $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_2 = \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_2$. By applying the overall induction hypothesis on $\mathsf{G}_0 \vdash \{\mathsf{T}^\#\}\ \mathsf{C}_0\ \{\mathsf{T}^\#\}$, we get $[\![\mathsf{C}_0]\!]\mathsf{T} \models \mathsf{T}^\#$; by applying the innermost induction hypothesis, we then get $\mathsf{f}[\![\mathsf{C}_0]\!]\mathsf{T} \models \mathsf{T}^\#$, so that $\mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_1 \overset{z}{=} \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_2$ which amounts to the desired $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1 \overset{z}{=} \mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_2$.

**not** ***true?***$([\![\mathsf{E}]\!]\mathsf{s}_1')$ **and not** ***true?***$([\![\mathsf{E}]\!]\mathsf{s}_2')$**.** Then $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1 = \mathsf{T}\ \mathsf{s}_1$ and $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_2 = \mathsf{T}\ \mathsf{s}_2$, and the claim is trivial.

***true?***$([\![\mathsf{E}]\!]\mathsf{s}_1')$ **but not** ***true?***$([\![\mathsf{E}]\!]\mathsf{s}_2')$**.** If $w \notin \mathsf{G}_0$, then by the logic, for all $x \in \mathrm{fv}(\mathsf{E})$, $[x \ltimes w] \in \mathsf{T}_0^\#$ implying $\mathsf{T}\ \mathsf{s}_1 \overset{x}{=} \mathsf{T}\ \mathsf{s}_2$, that is $\mathsf{s}_1'(x) = \mathsf{s}_2'(x)$. By Property 2.1 this implies $[\![\mathsf{E}]\!]\mathsf{s}_1' = [\![\mathsf{E}]\!]\mathsf{s}_2'$ – a contradiction. Thus $w \in \mathsf{G}_0$. Since $[z \ltimes w] \in \mathsf{T}^\#$, we infer from the logic that $z \neq \bot$. Thus, $z$ is a variable.

By Lemma 4.6 (applied to the sub-derivation), we infer that $z \notin \mathit{modified}(\mathsf{C}_0)$. We know that $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1 = \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_1$ and $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_2 = \mathsf{T}\ \mathsf{s}_2$. We can assume that $\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1 \neq \bot$, as otherwise the claim vacuously holds. By Lemma 2.3, $[\![\mathsf{C}_0]\!]\mathsf{T}\ \mathsf{s}_1\ z = \mathsf{T}\ \mathsf{s}_1\ z$, and, by the same lemma, $\mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\ \mathsf{s}_1\ z = [\![\mathsf{C}_0]\!]\mathsf{T}\ \mathsf{s}_1\ z$. So we have the desired relation:

$$\mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_1\ z = \mathsf{f}([\![\mathsf{C}_0]\!]\mathsf{T})\mathsf{s}_1\ z = \mathsf{T}\ \mathsf{s}_1\ z = \mathsf{T}\ \mathsf{s}_2\ z = \mathcal{F}\ \mathsf{f}\ \mathsf{T}\ \mathsf{s}_2\ z.$$

We have proved (1). We must now prove $\mathcal{P}([\![\mathsf{C}]\!])$, that is $\mathcal{P}(\sqcup_i \mathsf{f}_i)$. So assume that $\mathsf{T} \models \mathsf{T}^\#$ and that $[z \ltimes w] \in \mathsf{T}^\#$ and that $\mathsf{s}_1 \underset{w}{=} \mathsf{s}_2$; we must prove that $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}_1 \overset{z}{=} [\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}_2$. Clearly, there exists $\mathsf{j}$ such that $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}_1 = \mathsf{f}_\mathsf{j}\ \mathsf{T}\ \mathsf{s}_1$ and $[\![\mathsf{C}]\!]\mathsf{T}\ \mathsf{s}_2 = \mathsf{f}_\mathsf{j}\ \mathsf{T}\ \mathsf{s}_2$. The claim now follows from (1).

[Sub]. Assume that

$$\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$$

because with $\mathsf{G} \subseteq \mathsf{G}_0$ and $\mathsf{T}_0^\# \preceq \mathsf{T}_1^\#$ and $\mathsf{T}_2^\# \preceq \mathsf{T}^\#$ we have

$$\mathsf{G}_0 \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}\ \{\mathsf{T}_2^\#\}.$$

From our assumption $\mathsf{T} \models \mathsf{T}_0^\#$ we infer by Fact 3.4 that $\mathsf{T} \models \mathsf{T}_1^\#$, so inductively we can assume that $[\![\mathsf{C}]\!]\mathsf{T} \models \mathsf{T}_2^\#$. One more application of Fact 3.4 then yields the desired result.

## 6 An application of the logic: Forward Slicing

In this section we shall see how to compute the "low" forward slice of a program P. That is, we focus on one particular "high" variable $\mathsf{h}$, and eliminate all

$$[\mathsf{Assign}]\ \mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\, \mathtt{x} := \mathsf{E}\, \{\mathsf{T}^{\#}\} \quad \Rightarrow \quad \begin{cases} \mathtt{x} := \mathsf{E}, \text{ if } [\mathtt{x} \ltimes \mathtt{h}] \in \mathsf{T}^{\#} \\ \mathbf{skip}, \quad \text{if } [\mathtt{x} \ltimes \mathtt{h}] \notin \mathsf{T}^{\#} \end{cases}$$

$$[\mathsf{Seq}]\ \frac{\overbrace{\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\, \mathsf{C}_1\, \{\mathsf{T}_1^{\#}\}}^{\mathsf{D}_1} \quad \overbrace{\mathsf{G} \vdash \{\mathsf{T}_1^{\#}\}\, \mathsf{C}_2\, \{\mathsf{T}_2^{\#}\}}^{\mathsf{D}_2}}{\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\, \mathsf{C}_1 \,;\, \mathsf{C}_2\, \{\mathsf{T}_2^{\#}\}} \quad \Rightarrow \mathcal{S}(\mathsf{D}_1)\,;\mathcal{S}(\mathsf{D}_2)$$

$$[\mathsf{If}]\ \frac{\overbrace{\mathsf{G}_0 \vdash \{\mathsf{T}_0^{\#}\}\, \mathsf{C}_1\, \{\mathsf{T}^{\#}\}}^{\mathsf{D}_1} \quad \overbrace{\mathsf{G}_0 \vdash \{\mathsf{T}_0^{\#}\}\, \mathsf{C}_2\, \{\mathsf{T}^{\#}\}}^{\mathsf{D}_2}}{\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\, \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathsf{C}_1\ \mathbf{else}\ \mathsf{C}_2\, \{\mathsf{T}^{\#}\}} \quad \Rightarrow \quad \begin{cases} \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathcal{S}(\mathsf{D}_1)\ \mathbf{else}\ \mathcal{S}(\mathsf{D}_2), \text{ if } \mathtt{h} \notin \mathsf{G}_0 \\ \mathbf{skip}, \qquad \text{if } \mathtt{h} \in \mathsf{G}_0 \end{cases}$$

$$[\mathsf{While}]\ \frac{\overbrace{\mathsf{G}_0 \vdash \{\mathsf{T}^{\#}\}\, \mathsf{C}\, \{\mathsf{T}^{\#}\}}^{\mathsf{D}_0}}{\mathsf{G} \vdash \{\mathsf{T}^{\#}\}\, \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathsf{C}\, \{\mathsf{T}^{\#}\}} \quad \Rightarrow \quad \begin{cases} \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathcal{S}(\mathsf{D}_0), \text{ if } \mathtt{h} \notin \mathsf{G}_0 \\ \mathbf{skip}, \qquad \text{if } \mathtt{h} \in \mathsf{G}_0 \end{cases}$$

$$[\mathsf{Sub}]\ \frac{\overbrace{\mathsf{G}_1 \vdash \{\mathsf{T}_1^{\#}\}\, \mathsf{C}\, \{\mathsf{T}_2^{\#}\}}^{\mathsf{D}_0}}{\mathsf{G}_0 \vdash \{\mathsf{T}_0^{\#}\}\, \mathsf{C}\, \{\mathsf{T}_3^{\#}\}} \quad \Rightarrow \mathcal{S}(\mathsf{D}_0)$$

Fig. 4. Forward Slicing: the slicing function $\mathcal{S}$.

commands in $\mathsf{P}$ that may depend on $\mathtt{h}$, yielding a "slice" $\mathsf{P}'$. With $\mathtt{l}_1 \ldots \mathtt{l}_n$ the variables not depending on $\mathtt{h}$ in $\mathsf{P}$ (and thus considered "low"), our aim is that $\mathsf{P}'$ should be equivalent to $\mathsf{P}$ on $\mathtt{l}_1 \ldots \mathtt{l}_n$. Then a user, wanting to compute the low variables but (for security reasons) not given clearance to view $\mathtt{h}$, could be given $\mathsf{P}'$ to run, rather than $\mathsf{P}$.

The slicing function $\mathcal{S}$ is defined in Fig. 4, inductively on derivations in the logic (Fig. 3.) The idea is to replace by **skip** *(i)* assignments to variables that may depend on $\mathtt{h}$; *(ii)* conditionals and loops whose test may depend on $\mathtt{h}$.

With $s_1, s_2 \in \mathbf{Sto}_\perp$, we write $s_1 \stackrel{T^\#}{=} s_2$ to denote that $s_1 \stackrel{z}{=} s_2$ holds for all $z \in \mathbf{Var} \cup \{\perp\}$ such that $[z \ltimes h] \in T^\#$. We can now formulate correctness of slicing:

**Theorem 6.1** *Let* $D$ *be a derivation for the judgement* $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$*, and let* $\mathcal{S}(D) = C'$*. Then* $[\![C]\!]T\ s \stackrel{T^\#}{=} [\![C']\!]T\ s$.

In particular, if $T^\# = \{[l \ltimes h], [\perp \ltimes h]\}$ we can infer that either $[\![C]\!]T\ s$ and $[\![C']\!]T\ s$ are both $\perp$, or they are both $\neq \perp$ and agree on $l$.

On the other hand, if $T^\#$ contains only $[l \ltimes h]$ we can infer only partial correctness; it may happen that $[\![C]\!]T\ s = \perp$ but $[\![C']\!]T\ s \neq \perp$. For an example of that, consider the program **while** $h > 0$ **do skip**; it can be given a derivation

$$\frac{\{h\} \vdash \{\{[l \ltimes h], [h \ltimes l]\}\}\ \mathbf{skip}\ \{\{[l \ltimes h], [h \ltimes l]\}\}}{\emptyset \vdash \{\{[l \ltimes h], [h \ltimes l]\}\}\ \mathbf{while}\ h > 0\ \mathbf{do}\ \mathbf{skip}\ \{\{[l \ltimes h], [h \ltimes l]\}\}}$$

which by $\mathcal{S}$ is transformed into **skip**.

Theorem 6.1 follows immediately from the following main lemma, facilitating a proof by induction.

**Lemma 6.2** *Let* $D$ *be a derivation for the judgement* $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$*, and let* $\mathcal{S}(D) = C'$.
*If* $T_1\ s \stackrel{T_0^\#}{=} T_2\ s$ *then* $[\![C]\!]T_1\ s \stackrel{T^\#}{=} [\![C']\!]T_2\ s$.

**PROOF.** Induction in the derivation, where in all cases, we can assume that

$$T_1\ s \neq \perp \text{ and } T_2\ s \neq \perp.$$

For if that is not the case, then (by Fact 2.2) we have $[\![C]\!]T_1\ s = \perp$ or $[\![C']\!]T_2\ s = \perp$. We would therefore have the desired relation $[\![C]\!]T_1\ s \stackrel{z}{=} [\![C']\!]T_2\ s$ for all $z \in \mathbf{Var}$, but must still consider the case $[\perp \ltimes h] \in T^\#$. Then by Fact 4.4 we infer that $[\perp \ltimes h] \in T_0^\#$ and therefore $T_1\ s \stackrel{\perp}{=} T_2\ s$ implying $T_1\ s = T_2\ s = \perp$; by Fact 2.2 this implies $[\![C]\!]T_1\ s = [\![C']\!]T_2\ s = \perp$ and therefore the desired relation $[\![C]\!]T_1\ s \stackrel{\perp}{=} [\![C']\!]T_2\ s$.

We now embark on a case analysis.

[Assign]. Given $[z \ltimes h] \in T^\#$, we must show

$$[\![C]\!]T_1\ s \stackrel{z}{=} [\![C']\!]T_2\ s.$$

If $z = \bot$, the claim is trivial, since $[\![C]\!]T_1\ s \neq \bot$ and $[\![C']\!]T_2\ s \neq \bot$. So assume that $z \neq \bot$.

If $z \neq x$, then $[z \ltimes h] \in T_0^\#$, implying $T_1\ s\ z = T_2\ s\ z$. So no matter whether $C'$ is $x := E$ or **skip**, we have the desired equality

$$[\![C]\!]T_1\ s\ z = T_1\ s\ z = T_2\ s\ z = [\![C']\!]T_2\ s\ z.$$

Now assume that $z = x$, in which case Fig. 4 tells us that $C' = C$, and where the side conditions from Fig. 3 tell us that for all $y \in \mathrm{fv}(E)$, $[y \ltimes h] \in T_0^\#$ and thus $T_1\ s\ y = T_2\ s\ y$. Therefore, using Property 2.1, $[\![E]\!](T_1\ s) = [\![E]\!](T_2\ s)$. Again we therefore get the desired equality

$$[\![C]\!]T_1\ s\ z = [\![E]\!](T_1\ s) = [\![E]\!](T_2\ s) = [\![C']\!]T_2\ s\ z.$$

[Seq]. Assume that with $C$ of the form $C_1\ ; C_2$, the derivation $D$ of $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$ has two children: a derivation $D_1$ of a judgement $G \vdash \{T_0^\#\}\ C_1\ \{T_1^\#\}$, and a derivation $D_2$ of a judgement $G \vdash \{T_1^\#\}\ C_2\ \{T^\#\}$. With $C_1' = \mathcal{S}(D_1)$ and $C_2' = \mathcal{S}(D_2)$, we have $C' = \mathcal{S}(D) = C_1'\ ; C_2'$.

Inductively on $D_1$, we from $T_1\ s \overset{T_0^\#}{=} T_2\ s$ infer $[\![C_1]\!]T_1\ s \overset{T_1^\#}{=} [\![C_1']\!]T_2\ s$; inductively on $D_2$, we further infer $[\![C_2]\!]([\![C_1]\!]T_1)\ s \overset{T^\#}{=} [\![C_2']\!]([\![C_1']\!]T_2)\ s$ which amounts to the desired equality $[\![C]\!]T_1\ s \overset{T^\#}{=} [\![C']\!]T_2\ s$.

[Sub]. Assume that the derivation $D$ of $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$ has as child a derivation $D_0$ of a judgment $G_0 \vdash \{T_1^\#\}\ C\ \{T_2^\#\}$, where $G \subseteq G_0$ and $T_0^\# \preceq T_1^\#$ and $T_2^\# \preceq T^\#$. Here $C' = \mathcal{S}(D) = \mathcal{S}(D_0)$. Our assumption $T_1\ s \overset{T_0^\#}{=} T_2\ s$ clearly implies $T_1\ s \overset{T_1^\#}{=} T_2\ s$ (as $T_1^\# \subseteq T_0^\#$); inductively on $D_0$ we can thus infer that $[\![C]\!]T_1\ s \overset{T_2^\#}{=} [\![C']\!]T_2\ s$ which (as $T^\# \subseteq T_2^\#$) implies the desired equality $[\![C]\!]T_1\ s \overset{T^\#}{=} [\![C']\!]T_2\ s$.

[If]. Assume that with $C$ of the form **if** $E$ **then** $C_1$ **else** $C_2$, the derivation $D$ of $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$ has two children: a derivation $D_1$ of a judgement $G_0 \vdash \{T_0^\#\}\ C_1\ \{T^\#\}$, and a derivation $D_2$ of a judgement $G_0 \vdash \{T_0^\#\}\ C_2\ \{T^\#\}$. Let $C_1' = \mathcal{S}(D_1)$, and $C_2' = \mathcal{S}(D_2)$. Looking at Fig. 4, we see that there are two subcases:

$C' =$ **if** $E$ **then** $C_1'$ **else** $C_2'$**, with** $h \notin G_0$**.** The side condition in Fig. 3 tells us that for all $x \in \mathrm{fv}(E)$ it holds that $[x \ltimes h] \in T_0^\#$ and thus $T_1\ s\ x = T_2\ s\ x$. Therefore, using Property 2.1, $[\![E]\!](T_1\ s) = [\![E]\!](T_2\ s)$; we can assume wlog. that this value satisfies *true?*. Thus $[\![C]\!]T_1\ s = [\![C_1]\!]T_1\ s$ and $[\![C']\!]T_2\ s = [\![C_1']\!]T_2\ s$;

this amounts to the desired equality $[\![C]\!]T_1\ s \stackrel{T^\#}{=} [\![C']\!]T_2\ s$ since inductively on $D_1$ we infer that $[\![C_1]\!]T_1\ s \stackrel{T^\#}{=} [\![C_1']\!]T_2\ s$.

$C' = \textbf{skip, with } h \in G_0$**.** Let $[z \ltimes h] \in T^\#$ be given; we must show that

$$[\![C]\!]T_1\ s \stackrel{z}{=} [\![C']\!]T_2\ s.$$

If $z = \bot$, Lemma 4.5 (applied to $D_1$ and $D_2$) tells us that $[\![C_1]\!]T_1\ s \neq \bot$ and $[\![C_2]\!]T_1\ s \neq \bot$; thus $[\![C]\!]T_1\ s \neq \bot$, yielding the claim (since trivially, $[\![C']\!]T_2\ s \neq \bot$).

We thus now assume that $z \in \textbf{Var}$; from Lemma 4.6 (applied to $D_1$ and $D_2$) we then infer that

- $z \notin \mathit{modified}(C_1)$ and $z \notin \mathit{modified}(C_2)$, and therefore $z \notin \mathit{modified}(C)$ which by Lemma 2.3 implies that $[\![C]\!]$ preserves $z$;
- $[z \ltimes h] \in T_0^\#$, implying that $T_1\ s\ z = T_2\ s\ z$.

If $[\![C]\!]T_1\ s = \bot$, the claim is trivial; otherwise we from the above infer the desired equality:

$$[\![C]\!]T_1\ s\ z = T_1\ s\ z = T_2\ s\ z = [\![C']\!]T_2\ s\ z.$$

[While]. Assume that with $C$ of the form **while** $E$ **do** $C_0$, and with $T_0^\# = T^\#$, the derivation $D$ of $G \vdash \{T_0^\#\}\ C\ \{T^\#\}$ has one child: a derivation $D_0$ of a judgement $G_0 \vdash \{T^\#\}\ C_0\ \{T^\#\}$. Let $C_0' = \mathcal{S}(D_0)$. Looking at Fig. 4, we see that there are two subcases:

$C' = \textbf{while } E \textbf{ do } C_0'\textbf{, with } h \notin G_0$**.** Let $\mathcal{F}$ be the functor of $C$, let $\mathcal{F}'$ be the function of $C'$, let $f_i$ $(i \geq 0)$ be the iterands of $C$, and let $f_i'$ $(i \geq 0)$ be the iterands of $C'$. We shall prove by induction in $i$ that

$$\text{forall } T_1, T_2, s\text{: } T_1\ s \stackrel{T^\#}{=} T_2\ s \text{ implies } f_i\ T_1\ s \stackrel{T^\#}{=} f_i'\ T_2\ s. \tag{1}$$

In all cases, we can assume that $T_1\ s \neq \bot$ and $T_2\ s \neq \bot$. For if that is not the case, then (by Fact 2.2) we have $f_i\ T_1\ s = \bot$ or $f_i'\ T_2\ s = \bot$, which would imply the desired judgement, except if $[\bot \ltimes h] \in T^\#$ in which case we must show that $f_i\ T_1\ s = f_i'\ T_2\ s = \bot$ which (by Fact 2.2) can be done by showing $T_1\ s = T_2\ s = \bot$. But this follows from $T_1\ s \stackrel{\bot}{=} T_2\ s$.

For $i = 0$, the claim is obvious, since $\bot \stackrel{T^\#}{=} \bot$ always holds. Now consider the inductive step, where we must prove that $T_1\ s \stackrel{T^\#}{=} T_2\ s$ implies $\mathcal{F}\ f_i\ T_1\ s \stackrel{T^\#}{=} \mathcal{F}'\ f_i'\ T_2\ s$. Since $h \notin G_0$, the side condition in Fig. 3 tells us that for all $x \in \mathrm{fv(E)}$ it holds that $[x \ltimes h] \in T^\#$ and thus $T_1\ s\ x = T_2\ s\ x$. Therefore,

using Property 2.1, $[\![\mathsf{E}]\!](\mathsf{T}_1\ \mathsf{s}) = [\![\mathsf{E}]\!](\mathsf{T}_2\ \mathsf{s})$.

If this value does not satisfy *true?*, then $\mathcal{F}\ \mathsf{f}_\mathsf{i}\ \mathsf{T}_1\ \mathsf{s} = \mathsf{T}_1\ \mathsf{s}$ and $\mathcal{F}'\ \mathsf{f}'_\mathsf{i}\ \mathsf{T}_2\ \mathsf{s} = \mathsf{T}_2\ \mathsf{s}$, trivially implying the claim. Otherwise, our task is to prove that

$$\mathsf{f}_\mathsf{i}\ ([\![\mathsf{C}_0]\!]\mathsf{T}_1)\ \mathsf{s} \stackrel{\mathsf{T}^\#}{=} \mathsf{f}'_\mathsf{i}\ ([\![\mathsf{C}'_0]\!]\mathsf{T}_2)\ \mathsf{s}.$$

By applying the overall induction hypothesis on $\mathsf{D}_0$, we infer that $[\![\mathsf{C}_0]\!]\mathsf{T}_1\ \mathsf{s} \stackrel{\mathsf{T}^\#}{=} [\![\mathsf{C}'_0]\!]\mathsf{T}_2\ \mathsf{s}$; the claim then follows by applying the innermost induction hypothesis.

We can now return to our main task, trying to prove that $[\![\mathsf{C}]\!]\mathsf{T}_1\ \mathsf{s} \stackrel{\mathsf{T}^\#}{=} [\![\mathsf{C}']\!]\mathsf{T}_2\ \mathsf{s}$ (where we are given that $\mathsf{T}_1\ \mathsf{s} \stackrel{\mathsf{T}^\#_0}{=} \mathsf{T}_2\ \mathsf{s}$). But observe that there exists $\mathsf{j}$ such that $[\![\mathsf{C}]\!]\mathsf{T}_1\ \mathsf{s} = \mathsf{f}_\mathsf{j}\ \mathsf{T}_1\ \mathsf{s}$ and $[\![\mathsf{C}']\!]\mathsf{T}_2\ \mathsf{s} = \mathsf{f}'_\mathsf{j}\ \mathsf{T}_2\ \mathsf{s}$; therefore (1) yields the claim.

$\mathsf{C}' = \mathbf{skip,\ with}\ \mathsf{h} \in \mathsf{G}_0$**.** Let $[\mathsf{z} \ltimes \mathsf{h}] \in \mathsf{T}^\#$ be given; we must show that

$$[\![\mathsf{C}]\!]\mathsf{T}_1\ \mathsf{s} \stackrel{\mathsf{z}}{=} [\![\mathsf{C}']\!]\mathsf{T}_2\ \mathsf{s}.$$

The side condition in Fig. 3 tells us that $\mathsf{z} \in \mathbf{Var}$; from Lemma 4.6 (applied to $\mathsf{D}_0$) we then infer that $\mathsf{z} \notin \mathit{modified}(\mathsf{C}_0) = \mathit{modified}(\mathsf{C})$ which by Lemma 2.3 implies that $[\![\mathsf{C}]\!]$ preserves $\mathsf{z}$. If $[\![\mathsf{C}]\!]\mathsf{T}_1\ \mathsf{s} = \bot$, the claim is trivial; otherwise, the desired equality follows from the calculation

$$[\![\mathsf{C}]\!]\mathsf{T}_1\ \mathsf{s}\ \mathsf{z} = \mathsf{T}_1\ \mathsf{s}\ \mathsf{z} = \mathsf{T}_2\ \mathsf{s}\ \mathsf{z} = [\![\mathsf{C}']\!]\mathsf{T}_2\ \mathsf{s}\ \mathsf{z}$$

where the second equality follows from $[\mathsf{z} \ltimes \mathsf{h}] \in \mathsf{T}^\#$.

**Example 6.3** *(Illustrating forward slicing.)* For the program $\mathsf{l} := \mathsf{h}\ ;\mathsf{l} := 0$, the forward slice is $\mathbf{skip}\ ;\mathsf{l} := 0$, because we lose the independence $[\mathsf{l} \ltimes \mathsf{h}]$ after $\mathsf{l} := \mathsf{h}$. For the program $\mathsf{h} := \mathsf{l}\ ;\mathsf{l} := \mathsf{h}$, the forward slice is $\mathsf{h} := \mathsf{l}\ ;\mathsf{l} := \mathsf{h}$ itself, since we retain the independence $[\mathsf{l} \ltimes \mathsf{h}]$ after each of the commands in the sequence, and since the first command establishes the independence $[\mathsf{h} \ltimes \mathsf{h}]$.

## 7 Computing Independences

In Fig. 5 we define a function

$$\mathit{sp} : \mathbf{Context} \times \mathbf{Cmd} \times \mathbf{Independ} \to \mathbf{Independ}$$

with the intuition (formalized below) that given a control dependence $\mathsf{G}$, a command $\mathsf{C}$ and a precondition $\mathsf{T}^\#$, $\mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#)$ computes a postcondition

$T_1^\#$ such that $G \vdash \{T^\#\}\ C\ \{T_1^\#\}$ holds, and $T_1^\#$ is the "largest" set (wrt. the subset ordering) that makes the judgement hold. Thus we compute the "strongest provable postcondition", which might differ[4] from the strongest *semantic* postcondition, that is, the largest set $T_1^\#$ such that for all $T$, if $T \models T^\#$ then $[\![C]\!](T) \models T_1^\#$.

In a companion technical report [3], we show how to also compute "weakest precondition"; we conjecture that the development in Sect. 8 could alternatively be carried out using weakest precondition instead of strongest postcondition.

We now explain two of the cases in Fig. 5. In an assignment, $x := E$, the postcondition carries over all independences $[y \ltimes w]$ in the precondition if $y \neq x$; these independences are unaffected by the assignment to $x$. Suppose that $w$ does not occur in context $G$. Then $x$ is not control dependent on $w$. Moreover, if all variables referenced in $E$ are independent of $w$, then $[x \ltimes w]$ will be in the postcondition of the assignment.

The case for **while** is best explained by means of an example.

**Example 7.1** Consider the program

$$C = n := 0\ ;\mathbf{while}\ y > n\ \mathbf{do}\ l := x\ ;x := y\ ;y := h\ ;n := n + 1.$$

Let $T_0^\# \ldots T_{12}^\#$ be given by the following table, where $V$ denotes the set $\{h, l, n, x, y\}$. For reasons of space, we will represent a non-empty set as the concatenation of its elements in the table – thus $\{h, l, n, x\}$ is represented as $hlnx$. For example, the entry in the column for $T_6^\#$ and in the row for $x$ shows that $[x \ltimes h] \in T_6^\#$

[4] For example, let $C = l := h - h$ and $T^\# = \{[l \ltimes h]\}$. Then $[l \ltimes h]$ is in the strongest semantic postcondition, since for all $T$ and all $s_1, s_2$ we have $[\![C]\!]T\ s_1 \overset{l}{=} [\![C]\!]T\ s_2$ and therefore $[\![C]\!]T \models [l \ltimes h]$, but not in the strongest provable postcondition.

and $[\mathsf{x} \ltimes \mathsf{l}] \in \mathsf{T}_6^\#$ and $[\mathsf{x} \ltimes \mathsf{n}] \in \mathsf{T}_6^\#$.

| | $\mathsf{T}_0^\#$ | $\mathsf{T}_1^\#$ | $\mathsf{T}_2^\#$ | $\mathsf{T}_3^\#$ | $\mathsf{T}_4^\#$ | $\mathsf{T}_5^\#$ | $\mathsf{T}_6^\#$ | $\mathsf{T}_7^\#$ | $\mathsf{T}_8^\#$ | $\mathsf{T}_9^\#$ | $\mathsf{T}_{10}^\#$ | $\mathsf{T}_{11}^\#$ | $\mathsf{T}_{12}^\#$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathsf{h} \ltimes$ | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy | lnxy |
| $\mathsf{n} \ltimes$ | hlxy | V | V | V | V | hlnx | hlnx | hlnx | hlnx | lnx | lnx | lnx | lnx |
| $\mathsf{l} \ltimes$ | hnxy | hnxy | hln | hln | hln | hln | hn | ln | ln | ln | n | ln | ln |
| $\mathsf{x} \ltimes$ | hlny | hlny | hlny | hlnx | hlnx | hlnx | hln | hln | lnx | lnx | ln | ln | lnx |
| $\mathsf{y} \ltimes$ | hlnx | hlnx | hlnx | hlnx | lnx | lnx | lnx | lnx | lnx | lnx | lnx | lnx | lnx |
| $\bot \ltimes$ | V | V | V | V | V | V | hlnx | hlnx | hlnx | hlnx | lnx | lnx | lnx |

Our goal is to compute $sp(\emptyset, \mathsf{C}, \mathsf{T}_0^\#)$ and doing so involves the fixed point computation sketched below.

| | Iteration | | |
|---|---|---|---|
| | first | second | third |
| **while** $\mathsf{y} > \mathsf{n}$ **do** | $\mathsf{T}_1^\#$ | $\mathsf{T}_6^\#$ | $\mathsf{T}_{10}^\#$ |
| $\mathsf{G}_0$ : | $\{\mathsf{y}\}$ | $\{\mathsf{h}, \mathsf{y}\}$ | $\{\mathsf{h}, \mathsf{y}\}$ |
| $\mathsf{l} := \mathsf{x}$ | $\mathsf{T}_2^\#$ | $\mathsf{T}_7^\#$ | $\mathsf{T}_{11}^\#$ |
| $\mathsf{x} := \mathsf{y}$ | $\mathsf{T}_3^\#$ | $\mathsf{T}_8^\#$ | $\mathsf{T}_{12}^\#$ |
| $\mathsf{y} := \mathsf{h}$ | $\mathsf{T}_4^\#$ | $\mathsf{T}_8^\#$ | $\mathsf{T}_{12}^\#$ |
| $\mathsf{n} := \mathsf{n} + 1$ | $\mathsf{T}_5^\#$ | $\mathsf{T}_9^\#$ | $\mathsf{T}_{12}^\#$ |
| $\_ \cap \mathsf{T}_1^\# \setminus [\bot \ltimes \mathsf{G}_0]$ | $\mathsf{T}_6^\#$ | $\mathsf{T}_{10}^\#$ | $\mathsf{T}_{10}^\#$ |

For example, the entry $\mathsf{T}_9^\#$ in the column marked "second" and in the row marked "$\mathsf{n} := \mathsf{n} + 1$", denotes that $sp(\{\mathsf{h}, \mathsf{y}\}, \mathsf{n} := \mathsf{n} + 1, \mathsf{T}_8^\#) = \mathsf{T}_9^\#$.

Note that at the end of the first iteration, the independence set is $\mathsf{T}_6^\#$ and $[\mathsf{l} \ltimes \mathsf{h}]$ is still present; it takes a second iteration – the independence set is then $\mathsf{T}_{10}^\#$ – to filter $[\mathsf{l} \ltimes \mathsf{h}]$ out and thus detect insecurity. The third iteration affirms that $\mathsf{T}_{10}^\#$ is indeed a fixed point (of the functional $\mathcal{H}_{\mathbf{while}}^{\mathsf{T}_1^\#, \emptyset}$ defined in Fig. 5).

$$
\begin{array}{l}
sp(\mathsf{G}, \mathsf{x} := \mathsf{E}, \mathsf{T}_0^\#) = \\
\qquad \{[\mathsf{y} \ltimes w] \mid (\mathsf{y} \neq \mathsf{x} \vee \mathsf{y} = \bot) \wedge [\mathsf{y} \ltimes w] \in \mathsf{T}_0^\#\} \\
\quad \cup \quad \{[\mathsf{x} \ltimes w] \mid w \notin \mathsf{G} \wedge \forall \mathsf{y} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{y} \ltimes w] \in \mathsf{T}_0^\#\}
\end{array}
$$

$$
sp(\mathsf{G}, \mathsf{C}_1 \,;\, \mathsf{C}_2, \mathsf{T}_0^\#) = sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^\#))
$$

$$
\begin{array}{l}
sp(\mathsf{G}, \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2, \mathsf{T}_0^\#) = \\
\quad \text{let } \mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}_0^\#\} \\
\qquad \mathsf{T}_1^\# = sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}_0^\#) \\
\qquad \mathsf{T}_2^\# = sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}_0^\#) \\
\quad \text{in } \mathsf{T}_1^\# \cap \mathsf{T}_2^\#
\end{array}
$$

$$
\begin{array}{l}
sp(\mathsf{G}, \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0, \mathsf{T}_0^\#) = \\
\quad \text{let } \mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}} : \textbf{Independ} \to \textbf{Independ} \text{ be given by } (\mathsf{C} = \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0) \\
\qquad \mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}^\#) = \\
\qquad\quad \text{let } \mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}^\#\} \\
\qquad\quad \text{in } (sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}^\#) \cap \mathsf{T}_0^\#) \setminus \{[\bot \ltimes w] \mid w \in \mathsf{G}_0\} \\
\quad \text{in } lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}})
\end{array}
$$

Fig. 5. Strongest Postcondition.

Theorem 7.3 states the correctness of the function *sp*, that it indeed computes a postcondition. Then, Theorem 7.4 states that the postcondition computed by *sp* is the strongest postcondition. We shall rely on the following property:

**Lemma 7.2 (Monotonicity)** *For all* $\mathsf{C}$*, the following holds (for all* $\mathsf{G}$,$\mathsf{G}_1$,$\mathsf{T}^\#$,$\mathsf{T}_1^\#$*):*

*(1)* $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#)$ *is well-defined;*
*(2)* $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^\#, \mathsf{G}}$ *(when* $\mathsf{C}$ *is a* **while** *loop) is a monotone function;*
*(3) if* $\mathsf{G} \subseteq \mathsf{G}_1$ *then* $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \preceq sp(\mathsf{G}_1, \mathsf{C}, \mathsf{T}^\#)$*;*
*(4) if* $\mathsf{T}^\# \preceq \mathsf{T}_1^\#$ *then* $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \preceq sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_1^\#)$*.*

**PROOF.** Induction in $\mathsf{C}$, where the four parts of the lemma are proved simultaneously. We do a case analysis on $\mathsf{C}$; the only non-trivial case is where $\mathsf{C}$ is of the form **while** $\mathsf{E}$ **do** $\mathsf{C}_0$.

Using the induction hypothesis on $\mathsf{C}_0$, we infer that for all $\mathsf{T}_0^\#$, $\mathsf{G}$ it holds that $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}}$ is a monotone function on the complete lattice **Independ**. Hence $lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}})$, and thus $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#)$, is indeed well-defined.

Next assume that $\mathsf{T}^\# \preceq \mathsf{T}_1^\#$ and that $\mathsf{G} \subseteq \mathsf{G}_1$. Then clearly $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^\#, \mathsf{G}} \preceq \mathcal{H}_{\mathsf{C}}^{\mathsf{T}_1^\#, \mathsf{G}_1}$ (by the pointwise ordering) and therefore $lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^\#, \mathsf{G}}) \preceq lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}_1^\#, \mathsf{G}_1})$ which amounts to the desired relation $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \preceq sp(\mathsf{G}_1, \mathsf{C}, \mathsf{T}_1^\#)$.

**Theorem 7.3** *For all* $\mathsf{C}$, $\mathsf{G}$, $\mathsf{T}_0^\#$, *it holds that* $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#)\}$.

**PROOF.** Go by structural induction on $\mathsf{C}$; we perform a case analysis.

$\underline{\mathsf{C} = \mathsf{x} := \mathsf{E}.}$ Let $\mathsf{T}^\# = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#)$, and assume $[\mathsf{z} \ltimes \mathsf{w}] \in \mathsf{T}^\#$. There are two cases:

- if $\mathsf{z} \neq \mathsf{x}$ then $[\mathsf{z} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$.
- if $\mathsf{z} = \mathsf{x}$ then $\mathsf{w} \notin \mathsf{G}$, and $\forall \mathsf{y} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$.

This establishes $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$.

$\underline{\mathsf{C} = \mathsf{C}_1\,;\mathsf{C}_2.}$ Assume that $sp(\mathsf{G}, \mathsf{C}_1\,;\mathsf{C}_2, \mathsf{T}_0^\#) = \mathsf{T}^\#$ because $\mathsf{T}^\# = sp(\mathsf{G}, \mathsf{C}_2, \mathsf{T}_1^\#)$ where $\mathsf{T}_1^\# = sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^\#)$. By the induction hypothesis on $\mathsf{C}_1$ and on $\mathsf{C}_2$, we have

$$\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}_1^\#\} \text{ and } \mathsf{G} \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}_2\ \{\mathsf{T}^\#\}$$

from which we infer the desired relation $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\,;\mathsf{C}_2\ \{\mathsf{T}^\#\}$.

$\underline{\mathsf{C} = \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2.}$ Assume that $sp(\mathsf{G}, \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2, \mathsf{T}_0^\#) = \mathsf{T}^\#$ because $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}_0^\#\}$, $\mathsf{T}_1^\# = sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}_0^\#)$ and $\mathsf{T}_2^\# = sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}_0^\#)$ and $\mathsf{T}^\# = \mathsf{T}_1^\# \cap \mathsf{T}_2^\#$. Inductively, we have

$$\mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}_1^\#\} \text{ and } \mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_2\ \{\mathsf{T}_2^\#\}.$$

As $\mathsf{T}^\# \subseteq \mathsf{T}_1^\#$ and $\mathsf{T}^\# \subseteq \mathsf{T}_2^\#$ we have $\mathsf{T}_1^\# \preceq \mathsf{T}^\#$ and $\mathsf{T}_2^\# \preceq \mathsf{T}^\#$; by [Sub], this implies

$$\mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}^\#\} \text{ and } \mathsf{G}_0 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_2\ \{\mathsf{T}^\#\}.$$

This establishes the desired $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}$ **if** $\mathsf{E}$ **then** $\mathsf{C}_1$ **else** $\mathsf{C}_2\ \{\mathsf{T}^\#\}$, since $\mathsf{G} \subseteq \mathsf{G}_0$ and $\mathsf{w} \notin \mathsf{G}_0$ implies $\forall \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$.

$\underline{\mathsf{C} = \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0.}$ Assume that $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#) = \mathsf{T}^\#$ so we want to prove $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$. We have $\mathsf{T}^\# = lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}})$. By definition of a fixed point, $\mathsf{T}^\# = \mathcal{H}_{\mathsf{C}}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}^\#)$. Thus

$$\mathsf{T}^\# = (sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}^\#) \cap \mathsf{T}_0^\#) \setminus \{[\bot \ltimes \mathsf{w}] \mid \mathsf{w} \in \mathsf{G}_0\}$$

where $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}^\#\}$. Hence $sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}^\#) \preceq \mathsf{T}^\#$ and $\mathsf{T}_0^\# \preceq \mathsf{T}^\#$. We claim $\mathsf{G} \vdash \{\mathsf{T}^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$, which by [Sub] implies the desired $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$.

It remains to prove the claim, $\mathsf{G} \vdash \{\mathsf{T}^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$. By the induction hypothesis on $\mathsf{C}_0$, $\mathsf{G}_0 \vdash \{\mathsf{T}^{\#}\}\ \mathsf{C}_0\ \{sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}^{\#})\}$ and by [Sub] therefore

$$\mathsf{G}_0 \vdash \{\mathsf{T}^{\#}\}\ \mathsf{C}_0\ \{\mathsf{T}^{\#}\}.$$

Now we get $\mathsf{G} \vdash \{\mathsf{T}^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$ by an application of [While] because $\mathsf{G} \subseteq \mathsf{G}_0$, because $w \notin \mathsf{G}_0$ implies $\forall x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \in \mathsf{T}^{\#}$, and because if $w \in \mathsf{G}_0$ then $[\bot \ltimes w] \notin \mathsf{T}^{\#}$.

**Theorem 7.4** *For all judgements* $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$, $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^{\#}) \preceq \mathsf{T}^{\#}$.

**PROOF.** We perform induction in the derivation of $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$, and do a case analysis on the last rule applied:

[Sub]. Assume that $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$ because with $\mathsf{G} \subseteq \mathsf{G}_1$ and $\mathsf{T}_0^{\#} \preceq \mathsf{T}_2^{\#}$ and $\mathsf{T}_3^{\#} \preceq \mathsf{T}^{\#}$ we have $\mathsf{G}_1 \vdash \{\mathsf{T}_2^{\#}\}\ \mathsf{C}\ \{\mathsf{T}_3^{\#}\}$. Applying the induction hypothesis on that derivation, we get

$$sp(\mathsf{G}_1, \mathsf{C}, \mathsf{T}_2^{\#}) \preceq \mathsf{T}_3^{\#}$$

and by Lemma 7.2 we get $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^{\#}) \preceq sp(\mathsf{G}_1, \mathsf{C}, \mathsf{T}_2^{\#})$. This yields the desired relation

$$sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^{\#}) \preceq \mathsf{T}_3^{\#} \preceq \mathsf{T}^{\#}.$$

[Assign], with $\mathsf{C} = x := \mathrm{E}$. Assume that $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$, and let $\mathsf{T}_1^{\#} = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^{\#})$. We want $\mathsf{T}_1^{\#} \preceq \mathsf{T}^{\#}$. Accordingly, assume $[y \ltimes w] \in \mathsf{T}^{\#}$ to show $[y \ltimes w] \in \mathsf{T}_1^{\#}$. We have two cases:

- $x \neq y$. Then $[y \ltimes w] \in \mathsf{T}_0^{\#}$; hence $[y \ltimes w] \in \mathsf{T}_1^{\#}$ by the definition of *sp*.
- $x = y$. Then $w \notin \mathsf{G}$ and $\forall z \in \mathrm{fv}(\mathrm{E}) \bullet [z \ltimes w] \in \mathsf{T}_0^{\#}$; hence $[y \ltimes w] \in \mathsf{T}_1^{\#}$ by the definition of *sp*.

[Seq], with $\mathsf{C} = \mathsf{C}_1\,;\mathsf{C}_2$. Assume $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}\ \{\mathsf{T}^{\#}\}$ because $\mathsf{G} \vdash \{\mathsf{T}_0^{\#}\}\ \mathsf{C}_1\ \{\mathsf{T}_2^{\#}\}$ and $\mathsf{G} \vdash \{\mathsf{T}_2^{\#}\}\ \mathsf{C}_2\ \{\mathsf{T}^{\#}\}$. By the induction hypothesis on these derivations,

$$sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^{\#}) \preceq \mathsf{T}_2^{\#} \text{ and } sp(\mathsf{G}, \mathsf{C}_2, \mathsf{T}_2^{\#}) \preceq \mathsf{T}^{\#}$$

which by Lemma 7.2 enables us to infer that

$$sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^{\#})) \preceq \mathsf{T}^{\#}.$$

This is as desired, since $sp(\mathsf{G}, \mathsf{C}_1\,;\mathsf{C}_2, \mathsf{T}_0^{\#}) = sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^{\#}))$.

[If], with $\mathsf{C} = \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2$. Assume that $\mathsf{G} \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$ because $\mathsf{G}_1 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_1\ \{\mathsf{T}^\#\}$ and $\mathsf{G}_1 \vdash \{\mathsf{T}_0^\#\}\ \mathsf{C}_2\ \{\mathsf{T}^\#\}$ where $\mathsf{G} \subseteq \mathsf{G}_1$ and where $w \notin \mathsf{G}_1$ implies that $\forall x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \in \mathsf{T}_0^\#$. Inductively, via the judgements for $\mathsf{C}_1$ and $\mathsf{C}_2$, we obtain

$$sp(\mathsf{G}_1, \mathsf{C}_1, \mathsf{T}_0^\#) \preceq \mathsf{T}^\# \text{ and } sp(\mathsf{G}_1, \mathsf{C}_2, \mathsf{T}_0^\#) \preceq \mathsf{T}^\#.$$

Let $\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \notin \mathsf{T}_0^\#\}$. Note that $\mathsf{G}_0 \subseteq \mathsf{G}_1$. Thus by Lemma 7.2 we get

$$sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}_0^\#) \preceq \mathsf{T}^\# \text{ and } sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}_0^\#) \preceq \mathsf{T}^\#.$$

This yields the claim since $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#) = sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}_0^\#) \cap sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}_0^\#)$.

[While], with $\mathsf{C} = \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0$. Assume that $\mathsf{G} \vdash \{\mathsf{T}^\#\}\ \mathsf{C}\ \{\mathsf{T}^\#\}$ because $\mathsf{G}_1 \vdash \{\mathsf{T}^\#\}\ \mathsf{C}_0\ \{\mathsf{T}^\#\}$ where $\mathsf{G} \subseteq \mathsf{G}_1$, and where $w \notin \mathsf{G}_1$ implies that $\forall x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \in \mathsf{T}^\#$, and where $[\bot \ltimes w] \in \mathsf{T}^\#$ implies $w \notin \mathsf{G}_1$. Assume $\mathsf{T}_1^\# = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#)$ to show $\mathsf{T}_1^\# \preceq \mathsf{T}^\#$. By the definition of $sp$, $\mathsf{T}_1^\# = lfp(\mathcal{H}_\mathsf{C}^{\mathsf{T}^\#,\mathsf{G}})$. Let $\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists x \in \mathrm{fv}(\mathrm{E}) \bullet [x \ltimes w] \notin \mathsf{T}^\#\}$; then

$$\begin{aligned}\mathcal{H}_\mathsf{C}^{\mathsf{T}^\#,\mathsf{G}}(\mathsf{T}^\#) &= (sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}^\#) \cap \mathsf{T}^\#) \setminus \{[\bot \ltimes w] \mid w \in \mathsf{G}_0\} \\ &\supseteq (sp(\mathsf{G}_1, \mathsf{C}_0, \mathsf{T}^\#) \cap \mathsf{T}^\#) \setminus \{[\bot \ltimes w] \mid w \in \mathsf{G}_1\} = \mathsf{T}^\#\end{aligned}$$

Here the set inclusion follows from the observation that $\mathsf{G}_0 \subseteq \mathsf{G}_1$ and an application of Lemma 7.2; the last equality follows since $w \in \mathsf{G}_1$ implies $[\bot \ltimes w] \notin \mathsf{T}^\#$, and since the induction hypothesis tells us that $sp(\mathsf{G}_1, \mathsf{C}_0, \mathsf{T}^\#) \preceq \mathsf{T}^\#$.

This shows that $\mathcal{H}_\mathsf{C}^{\mathsf{T}^\#,\mathsf{G}}$ is reductive at $\mathsf{T}^\#$, that is $\mathcal{H}_\mathsf{C}^{\mathsf{T}^\#,\mathsf{G}}(\mathsf{T}^\#) \preceq \mathsf{T}^\#$, so by Tarski's theorem we infer the desired relation $\mathsf{T}_1^\# = lfp(\mathcal{H}_\mathsf{C}^{\mathsf{T}^\#,\mathsf{G}}) \preceq \mathsf{T}^\#$.

The following result, where we clearly cannot allow $y = \bot$, is useful for the developments in Sections 8:

**Lemma 7.5** *Given* $\mathsf{C}$*, and* $y \in \mathbf{Var}$ *with* $y \notin$ *modified*$(\mathsf{C})$*. Then for all* $\mathsf{T}_0^\#$*,* $\mathsf{G}$*,* $w$*:*

$$[y \ltimes w] \in \mathsf{T}_0^\# \textit{ implies } [y \ltimes w] \in sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#).$$

**PROOF.** Go by structural induction on $\mathsf{C}$; we perform a case analysis. In each case, our assumption is that $y \notin$ *modified*$(\mathsf{C})$ and that $[y \ltimes w] \in \mathsf{T}_0^\#$; we must show $[y \ltimes w] \in sp(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#)$.

$\mathsf{C} = x := \mathrm{E}$. Our assumptions imply that $y \neq x$, from which the result trivially follows.

$\underline{\mathsf{C} = \mathsf{C}_1 \,;\, \mathsf{C}_2.}$ Since $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_1)$ we can apply the induction hypothesis on $\mathsf{C}_1$, giving us $[\mathsf{y} \ltimes \mathsf{w}] \in \mathit{sp}(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^\#)$. Since $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_2)$ we can then apply the induction hypothesis on $\mathsf{C}_2$, giving us $[\mathsf{y} \ltimes \mathsf{w}] \in \mathit{sp}(\mathsf{G}, \mathsf{C}_2, \mathit{sp}(\mathsf{G}, \mathsf{C}_1, \mathsf{T}_0^\#))$. This yields the claim.

$\underline{\mathsf{C} = \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2.}$ Since $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_1)$ and $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_2)$, we can apply the induction hypothesis twice, yielding (using the terminology in Fig. 5)

$$[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_1^\# \text{ and } [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_2^\#.$$

Since $\mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#) = \mathsf{T}_1^\# \cap \mathsf{T}_2^\#$, this yields the claim.

$\underline{\mathsf{C} = \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0.}$ Let $\mathsf{T}^\# = \mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}_0^\#)$, then $\mathsf{T}^\# = \mathit{lfp}(\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}})$. Define

$$\mathsf{T}_1^\# = \mathsf{T}^\# \cup \{[\mathsf{y} \ltimes \mathsf{w}]\}$$

and let $\mathsf{G}_0$ be as in Fig. 5. By applying the induction hypothesis on $\mathsf{C}_0$ (possible since $\mathsf{y} \notin \mathit{modified}(\mathsf{C}_0)$) we get $[\mathsf{y} \ltimes \mathsf{w}] \in \mathit{sp}(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}_1^\#)$; since $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}_0^\#$ and $\mathsf{y} \neq \bot$ this implies

$$[\mathsf{y} \ltimes \mathsf{w}] \in \mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}_1^\#). \tag{1}$$

Since $\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}$ is a monotone function (by Lemma 7.2), we from $\mathsf{T}_1^\# \preceq \mathsf{T}^\#$ infer that $\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}_1^\#) \preceq \mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}^\#) = \mathsf{T}^\#$ and thus

$$\mathsf{T}^\# \subseteq \mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}_1^\#). \tag{2}$$

Combining (1) and (2), we infer $\mathsf{T}_1^\# \subseteq \mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}_1^\#)$, that is $\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}(\mathsf{T}_1^\#) \preceq \mathsf{T}_1^\#$. This shows that $\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}$ is reductive at $\mathsf{T}_1^\#$, so by Tarski's theorem we infer $\mathsf{T}^\# = \mathit{lfp}(\mathcal{H}_\mathsf{C}^{\mathsf{T}_0^\#, \mathsf{G}}) \preceq \mathsf{T}_1^\#$, that is $\mathsf{T}_1^\# \subseteq \mathsf{T}^\#$. This demonstrates that $[\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#$, as desired.

## 8 Modularity and the Frame Rule

Define $\mathit{lhs}(\mathsf{T}^\#) = \{\mathsf{y} \mid [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#\}$. Then we have (proof to follow shortly)

**Theorem 8.1 (Frame rule (I))** *Given* $\mathsf{T}_0^\#$ *and* $\mathsf{C}$. *Then for all* $\mathsf{T}^\#$, $\mathsf{G}$*:*

*(1) If* $\mathit{lhs}(\mathsf{T}_0^\#) \subseteq \mathbf{Var}$ *and* $\mathit{lhs}(\mathsf{T}_0^\#) \cap \mathit{modified}(\mathsf{C}) = \emptyset$
*then* $\mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) \supseteq \mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \cup \mathsf{T}_0^\#$.

*(2) If* $lhs(\mathsf{T}_0^\#) \subseteq \mathbf{Var}$ *and* $lhs(\mathsf{T}_0^\#) \cap \mathrm{fv}(\mathrm{C}) = \emptyset$
*then* $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \cup \mathsf{T}_0^\#$.

Note that the weaker premise in 1 does not imply the stronger consequence in 2, since (with $[\mathsf{z} \ltimes \mathsf{w}]$ playing the role of $\mathsf{T}_0^\#$)

$$sp(\emptyset, \mathsf{x} := \mathsf{y} + \mathsf{z}, \{[\mathsf{y} \ltimes \mathsf{w}]\} \cup \{[\mathsf{z} \ltimes \mathsf{w}]\}) = \{[\mathsf{y} \ltimes \mathsf{w}], [\mathsf{z} \ltimes \mathsf{w}], [\mathsf{x} \ltimes \mathsf{w}]\}$$
$$sp(\emptyset, \mathsf{x} := \mathsf{y} + \mathsf{z}, \{[\mathsf{y} \ltimes \mathsf{w}]\}) \cup \{[\mathsf{z} \ltimes \mathsf{w}]\} = \{[\mathsf{y} \ltimes \mathsf{w}], [\mathsf{z} \ltimes \mathsf{w}]\}.$$

In separation logic [18,26], the frame rule is motivated by the desire for local reasoning: if $\mathsf{C}_1$ and $\mathsf{C}_2$ modify disjoint regions of a heap, reasoning about $\mathsf{C}_1$ can be performed independently of the reasoning about $\mathsf{C}_2$. In our setting, a consequence of the frame rule is that when analyzing a command $\mathsf{C}$ occurring in a larger context, the relevant independences are the ones whose left hand sides occur in $\mathsf{C}$.

Theorem 8.1 is proved by observing that part (1) follows from Lemmas 7.5 and 7.2; then part (2) follows using the following result:

**Lemma 8.2** *Let* $\mathsf{T}_0^\#$ *and* $\mathsf{C}$ *be given, with* $lhs(\mathsf{T}_0^\#) \subseteq \mathbf{Var}$ *and* $lhs(\mathsf{T}_0^\#) \cap \mathrm{fv}(\mathrm{C}) = \emptyset$. *Then for all* $\mathsf{T}^\#$ *and* $\mathsf{G}$, $sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) \subseteq sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \cup \mathsf{T}_0^\#$.

**PROOF.** Go by structural induction on $\mathsf{C}$; we perform a case analysis.

$\underline{\mathsf{C} = \mathsf{x} := \mathsf{E}.}$ The claim follows from the following calculation, using that $lhs(\mathsf{T}_0^\#) \cap \mathrm{fv}(\mathrm{E}) = \emptyset$ and that $\mathsf{x} \notin lhs(\mathsf{T}_0^\#)$.

$$\begin{aligned}
& sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) \\
=& \{[\mathsf{y} \ltimes \mathsf{w}] \mid \mathsf{y} \neq \mathsf{x} \wedge [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\# \cup \mathsf{T}_0^\#\} \cup \\
& \{[\mathsf{x} \ltimes \mathsf{w}] \mid \mathsf{w} \notin \mathsf{G} \wedge \forall \mathsf{y} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\# \cup \mathsf{T}_0^\#\} \\
=& \{[\mathsf{y} \ltimes \mathsf{w}] \mid \mathsf{y} \neq \mathsf{x} \wedge [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\# \cup \mathsf{T}_0^\#\} \cup \\
& \{[\mathsf{x} \ltimes \mathsf{w}] \mid \mathsf{w} \notin \mathsf{G} \wedge \forall \mathsf{y} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#\} \\
=& \mathsf{T}_0^\# \cup \{[\mathsf{y} \ltimes \mathsf{w}] \mid \mathsf{y} \neq \mathsf{x} \wedge [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#\} \cup \\
& \{[\mathsf{x} \ltimes \mathsf{w}] \mid \mathsf{w} \notin \mathsf{G} \wedge \forall \mathsf{y} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{y} \ltimes \mathsf{w}] \in \mathsf{T}^\#\} \\
=& \mathsf{T}_0^\# \cup sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#)
\end{aligned}$$

$\underline{\mathsf{C} = \mathsf{C}_1 \,;\, \mathsf{C}_2.}$ Using our induction hypothesis and Lemma 7.2, we get

$$sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) = sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}^\# \cup \mathsf{T}_0^\#))$$

$$\subseteq sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}^\#) \cup \mathsf{T}_0^\#) \subseteq sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}^\#)) \cup \mathsf{T}_0^\#$$
$$= sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \cup \mathsf{T}_0^\#$$

$\underline{\mathsf{C} = \textbf{if } \mathsf{E} \textbf{ then } \mathsf{C}_1 \textbf{ else } \mathsf{C}_2.}$ Let

$$\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}^\# \cup \mathsf{T}_0^\#\}$$

where our assumptions imply that $lhs(\mathsf{T}_0^\#) \cap \mathrm{fv}(\mathrm{E}) = \emptyset$ and therefore also

$$\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}^\#\}.$$

Using the induction hypothesis, we now get

$$\begin{aligned} sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\# \cup \mathsf{T}_0^\#) &= sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^\# \cup \mathsf{T}_0^\#) \cap sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^\# \cup \mathsf{T}_0^\#) \\ &\subseteq (sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^\#) \cup \mathsf{T}_0^\#) \cap (sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^\#) \cup \mathsf{T}_0^\#) \\ &= (sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^\#) \cap sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^\#)) \cup \mathsf{T}_0^\# \\ &= sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^\#) \cup \mathsf{T}_0^\# \end{aligned}$$

$\underline{\mathsf{C} = \textbf{while } \mathsf{E} \textbf{ do } \mathsf{C}_0.}$ Let $\mathsf{H} = \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^\#, \mathsf{G}}$ and let $\mathsf{H}_0 = \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^\# \cup \mathsf{T}_0^\#, \mathsf{G}}$, we must show that $lfp(\mathsf{H}_0) \subseteq lfp(\mathsf{H}) \cup \mathsf{T}_0^\#$. Define $\mathsf{T}_1^\# = lfp(\mathsf{H}_0)$, and let

$$\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}_1^\# \setminus \mathsf{T}_0^\#\}$$

where our assumptions imply that $lhs(\mathsf{T}_0^\#) \cap \mathrm{fv}(\mathrm{E}) = \emptyset$ and therefore also

$$\mathsf{G}_0 = \mathsf{G} \cup \{w \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes w] \notin \mathsf{T}_1^\#\}.$$

Using Lemma 7.2 and our induction hypothesis on $\mathsf{C}_0$ we get (where $\mathsf{T}_\perp^\#$ denotes $\{[\perp \ltimes w] \mid w \in \mathsf{G}_0\}$)

$$\begin{aligned} \mathsf{T}_1^\# \setminus \mathsf{T}_0^\# &= \mathsf{H}_0(\mathsf{T}_1^\#) \setminus \mathsf{T}_0^\# \\ &= ((sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}_1^\#) \cap (\mathsf{T}^\# \cup \mathsf{T}_0^\#)) \setminus \mathsf{T}_\perp^\#) \setminus \mathsf{T}_0^\# \\ &= (sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}_1^\#) \cap \mathsf{T}^\#) \setminus \mathsf{T}_0^\# \setminus \mathsf{T}_\perp^\# \\ &\subseteq (sp(\mathsf{G}_0, \mathsf{C}_0, (\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#) \cup \mathsf{T}_0^\#) \cap \mathsf{T}^\#) \setminus \mathsf{T}_0^\# \setminus \mathsf{T}_\perp^\# \\ &\subseteq ((sp(\mathsf{G}_0, \mathsf{C}_0, (\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#)) \cup \mathsf{T}_0^\#) \cap \mathsf{T}^\#) \setminus \mathsf{T}_0^\# \setminus \mathsf{T}_\perp^\# \\ &\subseteq (sp(\mathsf{G}_0, \mathsf{C}_0, (\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#)) \cap \mathsf{T}^\#) \setminus \mathsf{T}_\perp^\# \\ &= \mathsf{H}(\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#) \end{aligned}$$

We have proved $\mathsf{H}(\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#) \preceq \mathsf{T}_1^\# \setminus \mathsf{T}_0^\#$. This shows that $\mathsf{H}$ is reductive at $\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#$, so by Tarski's theorem we infer $lfp(\mathsf{H}) \preceq \mathsf{T}_1^\# \setminus \mathsf{T}_0^\#$. This implies the

desired relation

$$\mathit{lfp}(\mathsf{H}_0) = \mathsf{T}_1^\# \subseteq (\mathsf{T}_1^\# \setminus \mathsf{T}_0^\#) \cup \mathsf{T}_0^\# \subseteq \mathit{lfp}(\mathsf{H}) \cup \mathsf{T}_0^\#.$$

As a consequence of Theorem 8.1, we get the following result:

**Corollary 8.3 (Frame rule (II))** *Assume that* $\mathsf{G} \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}\ \{\mathsf{T}_2^\#\}$ *and that* $\mathit{lhs}(\mathsf{T}_0^\#) \cap (\mathit{modified}(\mathsf{C}) \cup \{\bot\}) = \emptyset$. *Then* $\mathsf{G} \vdash \{\mathsf{T}_1^\# \cup \mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathsf{T}_2^\# \cup \mathsf{T}_0^\#\}$.

**PROOF.** Using Theorems 8.1 and 7.4 we get

$$\mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}_1^\# \cup \mathsf{T}_0^\#) \supseteq \mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}_1^\#) \cup \mathsf{T}_0^\# \supseteq \mathsf{T}_2^\# \cup \mathsf{T}_0^\#.$$

Since by Theorem 7.3 we have $\mathsf{G} \vdash \{\mathsf{T}_1^\# \cup \mathsf{T}_0^\#\}\ \mathsf{C}\ \{\mathit{sp}(\mathsf{G}, \mathsf{C}, \mathsf{T}_1^\# \cup \mathsf{T}_0^\#)\}$, the result follows by [Sub].

**Example 8.4** Assume that

$$\mathsf{G} \vdash \{\mathsf{T}_1^\#\}\ \mathsf{C}_1\ \{\mathsf{T}_3^\#\} \text{ and } \mathsf{G} \vdash \{\mathsf{T}_2^\#\}\ \mathsf{C}_2\ \{\mathsf{T}_4^\#\}.$$

Further assume that $\mathit{lhs}(\mathsf{T}_2^\#) \cap (\mathit{modified}(\mathsf{C}_1) \cup \{\bot\}) = \emptyset$ and that $\mathit{lhs}(\mathsf{T}_3^\#) \cap (\mathit{modified}(\mathsf{C}_2) \cup \{\bot\}) = \emptyset$. Then Corollary 8.3 yields

$$\mathsf{G} \vdash \{\mathsf{T}_1^\# \cup \mathsf{T}_2^\#\}\ \mathsf{C}_1\ \{\mathsf{T}_3^\# \cup \mathsf{T}_2^\#\} \text{ and } \mathsf{G} \vdash \{\mathsf{T}_3^\# \cup \mathsf{T}_2^\#\}\ \mathsf{C}_2\ \{\mathsf{T}_3^\# \cup \mathsf{T}_4^\#\}$$

and therefore $\mathsf{G} \vdash \{\mathsf{T}_1^\# \cup \mathsf{T}_2^\#\}\ \mathsf{C}_1\,;\mathsf{C}_2\ \{\mathsf{T}_3^\# \cup \mathsf{T}_4^\#\}$.

A traditional view of modularity in the security literature is the "hook-up property" [20]: if two programs are secure then their composition is secure as well. Our logic satisfies the hook-up property for sequential composition; in our context, a secure program is one which has $[\mathfrak{l} \ltimes \mathfrak{h}]$ as an invariant (if $[\mathfrak{l} \ltimes \mathfrak{h}]$ is in the precondition, it is also in the strongest postcondition). With this interpretation, Sabelfeld and Sands's hook-up theorem holds [29, Theorem 5].

## 9 The Smith-Volpano Security Type System

In the Smith-Volpano type system [31], variables are labelled by security types; for example, $\mathfrak{x} : (\mathsf{T}, \kappa)$ means that $\mathfrak{x}$ has type $\mathsf{T}$ and security level $\kappa$. The security typing rules are given in Fig. 6. To handle implicit flows due to conditionals, the technical development requires commands to be typed $(\mathbf{com}\,\kappa)$ with the intention that all variables assigned to in such commands have level

$$\frac{\Gamma, x : (T, \kappa) \vdash E : (T, \kappa)}{\Gamma, x : (T, \kappa) \vdash x := E : (\mathbf{com}\ \kappa)}$$

$$\frac{\Gamma \vdash E : (\mathbf{int}, \kappa) \qquad \Gamma \vdash C_1 : (\mathbf{com}\ \kappa) \qquad \Gamma \vdash C_2 : (\mathbf{com}\ \kappa)}{\Gamma \vdash \mathbf{if}\ E\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 : (\mathbf{com}\ \kappa)}$$

$$\frac{\Gamma \vdash C_1 : (\mathbf{com}\ \kappa) \qquad \Gamma \vdash C_2 : (\mathbf{com}\ \kappa)}{\Gamma \vdash C_1 ; C_2 : (\mathbf{com}\ \kappa)}$$

$$\frac{\Gamma \vdash E : (\mathbf{int}, \kappa) \qquad \Gamma \vdash C : (\mathbf{com}\ \kappa)}{\Gamma \vdash \mathbf{while}\ E\ \mathbf{do}\ C : (\mathbf{com}\ \kappa)} \qquad \frac{\Gamma \vdash C : (\mathbf{com}\ \kappa_1) \qquad \kappa \leq \kappa_1}{\Gamma \vdash C : (\mathbf{com}\ \kappa)}$$

Fig. 6. The Smith-Volpano Type System: Rules for Commands

at least $\kappa$. The judgement $\Gamma \vdash C : (\mathbf{com}\ \kappa)$ says that in the security type context $\Gamma$ that binds free variables in $C$ to security types, command $C$ has type $(\mathbf{com}\ \kappa)$.

We now show a conservative extension: if a program is well-typed in the Smith-Volpano system, then for any two traces, the current values of low variables are independent of the initial values of high variables. For simplicity, we consider a program with only two variables, $h$ with level $H$ and $l$ with level $L$.

**Theorem 9.1** *Assume that* $C$ *can be given a security type wrt. the environment* $h : (\_, H), l : (\_, L)$. *Then for all* $T^{\#}$, *if* $[l \ltimes h] \in T^{\#}$ *then* $[l \ltimes h] \in sp(\emptyset, C, T^{\#})$.

The upshot of the theorem is that a well-typed program has $[l \ltimes h]$ as *invariant*: if $[l \ltimes h]$ appears in the precondition, then it also appears in the strongest postcondition.

The theorem is a straightforward consequence of the following lemma which facilitates a proof by induction. For $L$ commands, the assumption $h \notin G$ in the lemma says that $L$ commands cannot be control dependent on $H$ guards.

**Lemma 9.2** *(1) Suppose* $h : (\_, H), l : (\_, L) \vdash C : (\mathbf{com}\ H)$. *Then for all* $G, T^{\#}$, *if* $[l \ltimes h] \in T^{\#}$ *then* $[l \ltimes h] \in sp(G, C, T^{\#})$.
*(2) Suppose* $h : (\_, H), l : (\_, L) \vdash C : (\mathbf{com}\ L)$. *Then for all* $G, T^{\#}$, *if* $[l \ltimes h] \in T^{\#}$ *and* $h \notin G$ *then* $[l \ltimes h] \in sp(G, C, T^{\#})$.

**PROOF.** We prove the two parts of the lemma in turn. In both cases we go

by induction on the derivation of $\mathsf{C}$ (in the system of Fig. 6), with cases on the last rule used.

(**Part 1**.)

$\underline{\mathsf{C} = \mathsf{z} := \mathsf{E}.}$ Clearly, $\mathsf{z} \neq \mathsf{l}$ as otherwise the assignment cannot be typed. By definition,

$$sp(\mathsf{G}, \mathsf{z} := \mathsf{E}, \mathsf{T}^{\#}) \supseteq \{[\mathsf{u} \ltimes \mathsf{w}] \mid \mathsf{u} \neq \mathsf{z} \wedge [\mathsf{u} \ltimes \mathsf{w}] \in \mathsf{T}^{\#}\} \ni [\mathsf{l} \ltimes \mathsf{h}].$$

$\underline{\mathsf{C} = \mathsf{C}_1 \,;\, \mathsf{C}_2.}$ We have $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C}_1 : (\mathbf{com}\ \mathsf{H})$ and $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C}_2 : (\mathbf{com}\ \mathsf{H})$. Inductively, we get

$$[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}^{\#}) \text{ and then } [\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}, \mathsf{C}_2, sp(\mathsf{G}, \mathsf{C}_1, \mathsf{T}^{\#})).$$

Thus $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}, \mathsf{C}_1 \,;\, \mathsf{C}_2, \mathsf{T}^{\#})$.

$\underline{\mathsf{C} = \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathsf{C}_1\ \mathbf{else}\ \mathsf{C}_2.}$ Then $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C}_1 : (\mathbf{com}\ \mathsf{H})$ and $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C}_2 : (\mathbf{com}\ \mathsf{H})$. Assume $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}^{\#}$. Inductively,

$$[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^{\#}) \text{ and } [\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^{\#})$$

where $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}^{\#}\}$. Thus $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^{\#}) \cap sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^{\#})$, and we are done.

$\underline{\mathsf{C} = \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathsf{C}_0.}$ Then $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C}_0 : (\mathbf{com}\ \mathsf{H})$. Let $\mathsf{T}_0^{\#} = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^{\#})$. Then $\mathsf{T}_0^{\#} = lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}})$. Hence $\mathsf{T}_0^{\#} = \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_0^{\#})$.

Let $\mathsf{T}_1^{\#} = \mathsf{T}_0^{\#} \cup [\mathsf{l} \ltimes \mathsf{h}]$. Now

$$\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#}) = (sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}_1^{\#}) \cap \mathsf{T}^{\#}) \setminus \{[\bot \ltimes \mathsf{w}] \mid \mathsf{w} \in \mathsf{G}_0\}$$

where $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}_1^{\#}\}$. Inductively, as $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}_1^{\#}$, we get $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_0, \mathsf{T}_1^{\#})$. Thus,

$$[\mathsf{l} \ltimes \mathsf{h}] \in \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#}). \tag{1}$$

Because $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}$ is a monotone function, from $\mathsf{T}_1^{\#} \preceq \mathsf{T}_0^{\#}$ we get $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#}) \preceq \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_0^{\#}) = \mathsf{T}_0^{\#}$. Thus

$$\mathsf{T}_0^{\#} \subseteq \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#}). \tag{2}$$

Combining (1) and (2), we get $\mathsf{T}_1^{\#} \subseteq \mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#})$, i.e., $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}(\mathsf{T}_1^{\#}) \preceq \mathsf{T}_1^{\#}$. This shows that $\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}$ is reductive at $\mathsf{T}_1^{\#}$, so by Tarski's theorem, $\mathsf{T}_0^{\#} = lfp(\mathcal{H}_{\mathsf{C}}^{\mathsf{T}^{\#}, \mathsf{G}}) \preceq \mathsf{T}_1^{\#}$, that is, $\mathsf{T}_1^{\#} \subseteq \mathsf{T}_0^{\#}$. Hence $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}_0^{\#}$.

Subtyping. For the subtyping rule, the result trivially follows by induction on the smaller derivation tree for $\mathsf{C}$.

This completes Part 1 of the proof.

(**Part 2.**)

Subtyping. Assume $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}^{\#}$ and $\mathsf{h} \notin \mathsf{G}$. By the typing rule, $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C} : (\mathbf{com}\ \mathsf{L})$ follows because $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{C} : (\mathbf{com}\ \kappa_1)$ for some $\kappa_1$. If $\kappa_1 = \mathsf{L}$, the result follows inductively. If $\kappa_1 = \mathsf{H}$, then we can apply Part 1 of the theorem to conclude that $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^{\#})$ holds for any $\mathsf{G}$; in particular, it must therefore hold for the $\mathsf{G}$ where $\mathsf{h} \notin \mathsf{G}$.

$\mathsf{C} = \mathsf{z} := \mathsf{E}$. Assume $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}^{\#}$ and $\mathsf{h} \notin \mathsf{G}$. By typing, $\mathsf{z} : (\_, \mathsf{L})$ and $\mathsf{h} \notin \mathrm{fv}(\mathrm{E})$, as otherwise the assignment cannot be typed. Hence $\mathsf{z} = \mathsf{l}$. By definition,[5]

$$sp(\mathsf{G}, \mathsf{z} := \mathsf{E}, \mathsf{T}^{\#}) \supseteq \{[\mathsf{z} \ltimes \mathsf{w}] \mid \mathsf{w} \notin \mathsf{G} \wedge \forall \mathsf{u} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{u} \ltimes \mathsf{w}] \in \mathsf{T}^{\#}\} \ni [\mathsf{l} \ltimes \mathsf{h}].$$

$\mathsf{C} = \mathsf{C}_1 ; \mathsf{C}_2$. Easy induction.

$\mathsf{C} = \mathbf{if}\ \mathsf{E}\ \mathbf{then}\ \mathsf{C}_1\ \mathbf{else}\ \mathsf{C}_2$. By typing, $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{E} : (\mathbf{int}, \mathsf{L})$. Hence $\mathsf{h} \notin \mathrm{fv}(\mathrm{E})$ and thus $\mathsf{h} \notin \mathsf{G}_0$, where $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}^{\#}\}$. Then inductively, we obtain $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^{\#})$ and $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^{\#})$. Hence $[\mathsf{l} \ltimes \mathsf{h}] \in sp(\mathsf{G}_0, \mathsf{C}_1, \mathsf{T}^{\#}) \cap sp(\mathsf{G}_0, \mathsf{C}_2, \mathsf{T}^{\#}) = sp(\mathsf{G}, \mathsf{C}, \mathsf{T}^{\#})$.

$\mathsf{C} = \mathbf{while}\ \mathsf{E}\ \mathbf{do}\ \mathsf{C}_0$. By typing, $\mathsf{h} : (\_, \mathsf{H}), \mathsf{l} : (\_, \mathsf{L}) \vdash \mathsf{E} : (\mathbf{int}, \mathsf{L})$. Hence $\mathsf{h} \notin \mathrm{fv}(\mathrm{E})$. Now the proof proceeds similarly to the corresponding case in Part 1 and is omitted, except that we note that to use the induction hypothesis on the derivation of $\mathsf{C}_0$, we need to show $\mathsf{h} \notin \mathsf{G}_0$, where $\mathsf{G}_0 = \mathsf{G} \cup \{\mathsf{w} \mid \exists \mathsf{x} \in \mathrm{fv}(\mathrm{E}) \bullet [\mathsf{x} \ltimes \mathsf{w}] \notin \mathsf{T}_1^{\#}\}$. Since $\mathsf{h} \notin \mathsf{G}$, it is sufficient to show that for all $\mathsf{x} \in \mathrm{fv}(\mathrm{E})$ it holds that $[\mathsf{x} \ltimes \mathsf{h}] \in \mathsf{T}_1^{\#}$. But this follows since $\mathsf{h} \notin \mathrm{fv}(\mathrm{E})$ and $[\mathsf{l} \ltimes \mathsf{h}] \in \mathsf{T}_1^{\#}$.

[5] In case there are several low variables, we would use the argument:

$$sp(\mathsf{G}, \mathsf{z} := \mathsf{E}, \mathsf{T}^{\#}) \supseteq \{[\mathsf{u} \ltimes \mathsf{w}] \mid \mathsf{u} \neq \mathsf{z} \wedge [\mathsf{u} \ltimes \mathsf{w}] \in \mathsf{T}^{\#}\} \ni [\mathsf{l} \ltimes \mathsf{h}]$$

to deal with low variables not assigned to.

## 10 Discussion

**Perspective.** This paper specifies an information flow analysis for confidentiality using a Hoare-like logic and shows an application of the analysis to a program transformation, namely, slicing. The concrete pre- and post-traces of a program are abstracted by independences. Independences can be statically checked against the logic and can be statically computed using strongest post-conditions. We also show how the notion of independences underlies a classic type-based information flow analysis due to Smith and Volpano [31].

Giacobazzi and Mastroeni [14] consider attackers as abstract interpretations and generalize the notion of noninterference by parameterizing it wrt. what an attacker can analyze about the input/output information flow. For instance, assume an attacker can only analyze the *parity* (odd/even) of values. Then

$$\textbf{while } \mathsf{h} \textbf{ do } \mathsf{l} := \mathsf{l} + 2 \,;\mathsf{h} := \mathsf{h} - 1$$

is secure, although it contains an update of a low variable under a high guard. We might try to model this approach in our framework by parameterizing Definition 3.1 wrt. parity, but it is not clear how to alter the proof rules accordingly. Instead, we envision our logic to be put on top of abstract interpretations. In the parity example, the above program would be abstracted to

$$\textbf{while } \mathsf{h} \textbf{ do } \mathsf{h} := \mathsf{h} - 1$$

which our logic already deems secure for an attacker not able to observe nontermination.

**Related work.** The most closely related work is that of Clark, Hankin, and Hunt [8], who consider a language similar to ours and then extend it to Idealized Algol, requiring distinguishing between identifiers and locations. The analysis for Idealized Algol is split in two stages: the first stage does a control-flow analysis, specified using a flow logic [21]. The second stage specifies what is an acceptable information flow analysis with respect to the control-flow analysis. The precision of the control-flow analysis influences the precision of the information flow analysis. Flow logics usually do not come with a frame rule so it is unclear what modularity properties their analysis satisfies. For each statement $\mathsf{S}$ in the program, they compute the set of dependences introduced by $\mathsf{S}$; a pair $(\mathsf{x}, \mathsf{y})$ is in that set if different values for $\mathsf{y}$ prior to execution of $\mathsf{S}$ may result in different values for $\mathsf{x}$ after execution of $\mathsf{S}$. For a complete program, they thus, as expected, compute essentially the same information as we do, but the information computed *locally* is different from ours: we estimate

if different *initial* values of $\mathsf{y}$, i.e., values of $\mathsf{y}$ prior to execution of *the whole program*, may result in different values for $\mathsf{x}$ after execution of $\mathsf{S}$.

Joshi and Leino [19] provide an elegant semantic characterization of noninterference that allows handling both nontermination-sensitive and nontermination-insensitive noninterference. Their notion of security for a command $\mathsf{C}$ is equationally characterized by $\mathsf{C}\,;\mathsf{HH} = \mathsf{HH}\,;\mathsf{C}\,;\mathsf{HH}$, where $\mathsf{HH}$ means that an arbitrary value is assigned to a high variable. They show how to express their notion of security in Dijkstra's weakest precondition calculus. Although they do not consider synthesizing loop invariants, this can certainly be done via a fixpoint computation with weakest preconditions. However, their work is not concerned with computing dependences.

Darvas, Hähnle and Sands [12] use dynamic logic to express secure information flow in JavaCard. They discuss several ways that noninterference can be expressed in a program logic, one of which is as follows: consider a program with variables $\mathsf{l}$ and $\mathsf{h}$. Consider another copy of the program with $\mathsf{l}$, $\mathsf{h}$ relabeled to fresh variables $\mathsf{l}'$, $\mathsf{h}'$ respectively. Then, noninterference holds in the following situation: running the original program and the copy sequentially such that the initial state satisfies $\mathsf{l} = \mathsf{l}'$ should yield a final state satisfying $\mathsf{l} = \mathsf{l}'$. They are also interested in showing insecurity, by exhibiting distinct initial values for high variables that give distinct current values of low variables; to achieve this accuracy, they need the power of a general purpose theorem prover, which is also helpful in that they can express declassification, as well as treat exceptions (which most approaches based on static analysis cannot easily be extended to deal with).

Barthe, D'Argenio and Rezk [6] use the same idea of self-composition (i.e., composing a program with a copy of itself) as Darvas et alii and investigate "abstract" noninterference [14] for several languages. By parameterizing noninterference with a property, they are able to handle more general information flow policies, including a form of declassification known as delimited information release [27]. They show how self-composition can be formulated in logics describing these languages, namely, Hoare logic, separation logic, linear temporal logic, etc. They also discuss how to use their results for model checking programs with finite state spaces to check satisfaction of their generalized definition of noninterference.

The first work that used a Hoare-style semantics to reason about information flow was by Andrews and Reitman [4]. Their assertions keep track of the security level of variables, and are able to deal even with parallel programs. However, no formal correctness result is stated.

**Conclusion.** The work reported in this paper was inspired in part by presentations by Roberto Giacobazzi and Reiner Hähnle at the Dagstuhl Seminar on Language-based Security (October 2003). The reported work is only the first step in our goal to formulate more general definitions of noninterference in terms of program independence, such that the definitions support modular reasoning. We are in the process of extending the framework in this paper to handle a richer language, with methods, pointers, objects and dynamic memory allocation. An obvious goal is interprocedural reasoning about variable and field independences, perhaps using a higher-order version of the frame rule [23]. We would also like to explore, via abstract interpretation and perhaps following Schmidt's development [30], whether our Hoare-like logic is the "best" possible one with respect to the underlying abstract interpretation.

**Acknowledgements.** We thank Reiner Hähnle, Peter O'Hearn, Tamara Rezk, David Sands, and Hongseok Yang, as well as the participants of the *Open Software Quality* meeting in Santa Cruz, May 2004, and the anonymous reviewers of Static Analysis Symposium (SAS) 2004, for useful comments on a draft of this paper. Thanks to Sruthi Bandhakavi for implementing the analysis reported in this paper and for her help with Example 7.1. Thanks to David Schmidt for many discussions on abstract interpretation. Finally, thanks are due to Roberto Giacobazzi for organizing a very stimulating SAS 2004, and to the city of Verona for its warmth and its ice creams.

## References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 147–160, 1999.

[2] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. In *Static Analysis Symposium (SAS)*, volume 3148 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 2004.

[3] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. Technical Report CIS TR 2004-3, Kansas State University, April 2004.

[4] G. R. Andrews and R. P. Reitman. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56–75, January 1980.

[5] Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 253–270. IEEE Computer Society Press, 2002.

[6] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society Press, 2004.

[7] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corp., 1973.

[8] David Clark, Chris Hankin, and Sebastian Hunt. Information flow for Algol-like languages. *Computer Languages*, 28(1):3–28, 2002.

[9] Ellis S. Cohen. Information transmission in sequential programs. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 297–335. Academic Press, 1978.

[10] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 238–252. ACM Press, New York, NY, 1977.

[11] Patrick Cousot and Radhia Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. In *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages, SIGPLAN Notices*, volume 12, pages 1–12. ACM Press, August 1977.

[12] Ádám Darvas, Reiner Hähnle, and Dave Sands. A theorem proving approach to analysis of secure information flow. Technical Report 2004-01, Department of Computing Science, Chalmers University of Technology and Göteborg University, 2004. A fuller version of a paper appearing in Workshop on Issues in the Theory of Security, 2003.

[13] Dorothy Denning and Peter Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

[14] Roberto Giacobazzi and Isabella Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 186–197, 2004.

[15] J. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20, 1982.

[16] Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–377, 1998.

[17] Sebastian Hunt and David Sands. Binding time analysis: A new PERspective. In *Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, volume 26 (9) of *SIGPLAN Notices*, pages 154–165, 1991.

[18] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 14–26, 2001.

[19] Rajeev Joshi and K. Rustan M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.

[20] Daryl McCullough. Specifications for multi-level security and a hook-up. In *IEEE Symposium on Security and Privacy*, pages 161–166, 1987.

[21] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999. Web page at `www.imm.dtu.dk/~riis/PPA/ppa.html`.

[22] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2001.

[23] Peter O'Hearn, Hongseok Yang, and John Reynolds. Separation and information hiding. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 268–280, 2004.

[24] Peter Ørbæk and Jens Palsberg. Trust in the λ-calculus. *Journal of Functional Programming*, 7(6):557–591, November 1997.

[25] François Pottier and Vincent Simonet. Information flow inference for ML. *ACM Transactions on Programming Languages and Systems*, 25(1):117–158, January 2003.

[26] John C. Reynolds. Separation logic: a logic for shared mutable data structures. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 55–74. IEEE Computer Society Press, 2002.

[27] Andrei Sabelfeld and Andrew Myers. A model for delimited information release. In *Proceedings of the International Symposium on Software Security (ISSS)*, number 3233 in Lecture Notes in Computer Science, pages 174–191. Springer-Verlag, 2004.

[28] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[29] Andrei Sabelfeld and David Sands. A Per model of secure information flow in sequential programs. *Higher-order and Symbolic Computation*, 14(1):59–91, 2001.

[30] David A. Schmidt. Structure-preserving binary relations for program abstraction. In *The Essence of Computation: Complexity, Analysis, Transformation – Essays dedicated to Neil D. Jones*, number 2566 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[31] Dennis Volpano and Geoffrey Smith. A type-based approach to program security. In *Proceedings of TAPSOFT'97*, number 1214 in Lecture Notes in Computer Science, pages 607–621. Springer-Verlag, 1997.