LANGUAGE-BASED INFORMATION FLOW BASICS

ASLAN ASKAROV

Introduction

These notes cover basics of language-based information flow security.[1] We consider a toy imperative language, define its semantics using small-step operational semantics, use the semantics to introduce the basic notions of noninterference, and illustrate the ideas behind type-based enforcement of information flow.

Comments, questions, errors, etc, are all welcome – please email them to `aslan@cs.au.dk`.

1. Small imperative language

Consider simple imperative language given by the following grammar.

$$c ::= \mathsf{skip} \mid x := e \mid c; c \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c \mid \mathsf{while}\ e\ \mathsf{do}\ c$$
$$e ::= n \mid x \mid e\ op\ e$$

Assume that *op* ranges over total operations on arithmetic expressions. Consider memories to be total functions from variable names to values. We use a big-step evaluation relation for expressions, defined in Figure 1.

For commands, we introduce an auxiliary command stop that designates terminal configurations. Semantics for commands is given using a small-step relation $\langle c, m\rangle \to \langle c', m'\rangle$, where c is the starting command, m is the starting memory, c' is the updated command or stop, and m' is the updated memory. We refer to the pair $\langle c, m\rangle$ as a *semantic configuration*.

Rules for the semantic transitions are given in Figure 2. where, $m[x \mapsto v]$ is a shorthand for *memory update*:

$$m[x \mapsto v] \triangleq \lambda y\ .\ \text{if } x = y \text{ then } v \text{ else } m(y)$$

We write $\langle c, m\rangle \to^* \langle c', m'\rangle$ when configuration $\langle c, m\rangle$ can reach configuration $\langle c', m'\rangle$ in zero or many steps.

2. Noninterference

Figure 3 depicts an intuition for noninterference.

Date: June 26, 2017.

[1]These notes are developed in the context of the Language-Based Security course at Aarhus University.

$$\text{E-CONST}\ \frac{}{\langle n, m\rangle \Downarrow n} \qquad \text{E-VAR}\ \frac{m(x) = v}{\langle x, m\rangle \Downarrow v} \qquad \text{E-OP}\ \frac{\langle e_i, m\rangle \Downarrow v_i, i = 1, 2 \qquad v = v_1\ \mathbf{op}\ v_2}{\langle e_1\ op\ e_2, m\rangle \Downarrow v}$$

Figure 1. Semantics of expressions

$$\text{S-SKIP}\quad \frac{}{\langle \mathsf{skip}, m\rangle \to \langle \mathsf{stop}, m\rangle} \qquad \text{S-ASSIGN}\quad \frac{\langle e, m\rangle \Downarrow v}{\langle x := e, m\rangle \to \langle \mathsf{stop}, m[x \mapsto v]\rangle} \qquad \text{S-SEQ1}\quad \frac{\langle c_1, m\rangle \to \langle \mathsf{stop}, m'\rangle}{\langle c_1; c_2, m\rangle \to \langle c_2, m'\rangle}$$

$$\text{S-SEQ2}\quad \frac{\langle c_1, m\rangle \to \langle c_1', m'\rangle \qquad c_1' \neq \mathsf{stop}}{\langle c_1; c_2, m\rangle \to \langle c_1; c_2, m'\rangle} \qquad \text{S-IF1}\quad \frac{\langle e, m\rangle \Downarrow v \qquad v \neq 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, m\rangle \to \langle c_1, m\rangle}$$

$$\text{S-IF2}\quad \frac{\langle e, m\rangle \Downarrow v \qquad v = 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, m\rangle \to \langle c_2, m\rangle} \qquad \text{S-WHILE}\quad \frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ c, m\rangle \to \langle \mathsf{if}\ e\ \mathsf{then}\ c; \mathsf{while}\ e\ \mathsf{do}\ c\ \mathsf{else}\ \mathsf{skip}, m\rangle}$$

FIGURE 2. Semantics of commands

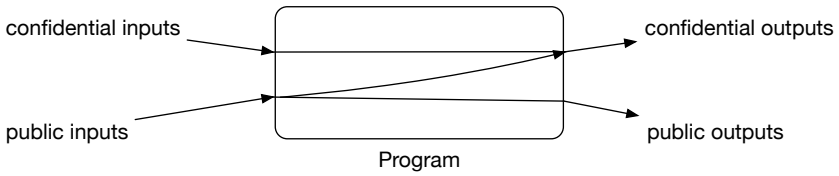

FIGURE 3. Intuition for noninterference

2.1. **Basic end-to-end definition.** Assume that the set of variables **Vars** is partitioned into two disjoint sets – the set **PubVars** of public variables, and the set **SecVars** of secret variables.

$$\mathbf{Vars} = \mathbf{PubVars} \uplus \mathbf{SecVars}$$

Definition 1 (Memory agreement on public variables)**.** Given two memories, m_1 and m_2, say that they agree on public variables, written $m_1 \sim m_2$, when

$$m_1 \sim m_2 \triangleq \forall x \in \mathbf{PubVars}\ .\ m_1(x) = m_2(x)$$

Definition 2 (Basic noninterference)**.** Program c is secure when for all pairs of memories m_1 and m_2 such that $m_1 \sim m_2$, and

$$\langle c, m_1\rangle \to^* \langle \mathsf{stop}, m_1'\rangle$$

and

$$\langle c, m_2\rangle \to^* \langle \mathsf{stop}, m_2'\rangle$$

it holds that $m_1' \sim m_2'$.

2.2. **Examples.** In the following examples, for clarity, we assume that variables that have suffix `_p` are public, and variables that have suffix `_s` are secret.

Example 1. Program

```
x_p := y_s
```

is insecure according to Definition 2.

Example 2. Program

```
x_p := 42
```

is secure according to Definition 2.

Example 3. Program

```
y_s := 42; x_p := y_s
```

is secure according to Definition 2.

Example 4. Program

```
if y_s > 0
   then x_p := 1
   else x_p := 0
```

is insecure according to Definition 2.

Example 5. Program

```
if y_s > 0
   then x_p := 0
   else x_p := 0
```

is secure according to Definition 2.

Exercise 1. Explain why the above example programs are secure or insecure (w.r.t. Definition 2)

Exercise 2. What can be said about security of the following program?

```
while y_s > 0
   do skip
```

Exercise 3. Definition 2 is *termination-insensitive*. What would a termination-sensitive variation of this definition – that is a definition that would rule out programs that leak by their termination behavior as insecure – look like? How would it change security of the above examples?

Exercise 4. Note that we were very implicit about *attacker model* so far. What *attacker model* do we actually assume in this section? For example, consider program

```
x_p := y_s; x_p := 0
```

Is this program accepted by Definition 2? Is our current attacker model realistic? What alternative attacker models can you propose?

Exercise 5 (*)**.** Consider an extension of our programming language with a construct for *nondeterministic assignment* $x := \mathsf{nondet}()$. Formally, we extend the language so that the extended grammar is now

$$c ::= \ldots \mid x := \mathsf{nondet}()$$

The operational semantics for the new construct is given by the following rule

$$\frac{}{\langle x := \mathsf{nondet}(), m\rangle \rightarrow \langle x := v, m\rangle}$$

Here, v is an arbitrary value (in other words, the semantics of the language is now nondeterministic)[2]. Observe that, with the addition of this construct, it is difficult to justify applicability of Definition 2. For example, a program such as `x_p` := $\mathsf{nondet}()$ always terminates, but because the final value of `x_p` may be different, this program is rejected by Definition 2, yet clearly, the program is innocuous.

Extend Definition 2 so that it can be justified for programs written with nondeterministic assignments.

Hint. What does your extended definition say about security of the following programs

[2]This may model a random assignment primitive in a mainstream language; it can also model reading information from a nondeterministic input source, such as user input or system clock.

(1)
```
x_p := nondet();
if z_s <= x_p then y_p := 0
              else y_p := 1
```

(2)
```
x_p := nondet();
if x_p then y_p := z_s
       else y_p := 42
```

(3)
```
x_p := nondet();
if x_p - x_p then y_p := z_s
             else y_p := 42
```

(4)
```
x_p := nondet();
x2_p := nondet();
if x_p - x2_p then y_p := z_s
              else y_p := 42
```

Remarks. Structural Operational Semantics has been introduced by Plotkin [Plotkin(1981)]; for a nice exposition, see Winskell's text [Winskel(1993)]. The idea of noninterference is introduced in [Goguen and Meseguer(1982)]. Sabelfeld and Myers [Sabelfeld and Myers(2003)] survey many of the basic approaches of language-based information-flow security.

$$\text{T-INT}\ \frac{}{\Gamma \vdash n : \ell} \qquad \text{T-VAR}\ \frac{\Gamma(x) = \ell}{\Gamma \vdash x : \ell} \qquad \text{T-OP}\ \frac{\Gamma \vdash e_1 : \ell_i, (i = 1, 2)}{\Gamma \vdash e_1\ op\ e_2 : \ell_1 \sqcup \ell_2}$$

FIGURE 4. Security typing of expressions

$$\text{T-SKIP}\ \frac{}{\Gamma, pc \vdash \mathsf{skip}} \qquad \text{T-ASSIGN}\ \frac{\Gamma \vdash e : \ell \qquad pc \sqcup \ell \sqsubseteq \Gamma(x)}{\Gamma, pc \vdash x := e} \qquad \text{T-SEQ}\ \frac{\Gamma, pc \vdash c_i, (i = 1, 2)}{\Gamma, pc \vdash c_1; c_2}$$

$$\text{T-IF}\ \frac{\Gamma \vdash e : \ell \qquad \Gamma, pc \sqcup \ell \vdash c_i, (i = 1, 2)}{\Gamma, pc \vdash \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2} \qquad \text{T-WHILE}\ \frac{\Gamma \vdash e : \ell \qquad \Gamma, pc \sqcup \ell \vdash c}{\Gamma, pc \vdash \mathsf{while}\ e\ \mathsf{do}\ c}$$

3. Type-based enforcement

One way to enforce noninterference is to do it statically using *security types*. We introduce a variable security type environment, denoted as Γ, that maps variable names to *security levels*. We start with a simplifying assumption that security levels, denoted by ℓ, can be either Public or Secret.

Typing rules for expressions have form $\Gamma \vdash e : \ell$. Security typing rules for expressions are given in Figure 4. In these rules, $\ell_1 \sqcup \ell_2$ corresponds to the *least upper bounds* of the two security levels ℓ_1 and ℓ_2. In our setting, we let $\ell_1 \sqcup \ell_2 = \mathsf{Public}$ when both ℓ_1 and ℓ_2 are Public; otherwise $\ell_1 \sqcup \ell_2 = \mathsf{Secret}$.

Exercise 6. What is the intuition behind the rule T-OP?

For commands, we use a security typing judgment that has the form $\Gamma, pc \vdash c$, where pc is the *program counter level* – it specifies how much information is leaked by knowing that a particular program point in the program is reached.

In rule T-ASSIGN, we use label comparison operator $\sqsubseteq$. For our setting, we have that

$$\ell \sqsubseteq \ell \qquad \text{and} \qquad \mathsf{Public} \sqsubseteq \mathsf{Secret}$$

Example derivation tree.

Suppose we have a program

```
x_p := 0;
while (y_s) do y_s := y_s - 1
x_p := 1;
```

Figure 3 gives an example derivation tree for this program (depending on how we associate the sequential composition), assuming an environment Γ such that $\Gamma(\texttt{x\_p}) = \mathsf{Public}$ and $\Gamma(\texttt{y\_s}) = \mathsf{Secret}$.

$$\dfrac{\dfrac{\Gamma \vdash 0 : \mathsf{Public} \qquad \mathsf{Public} \sqsubseteq \Gamma(\mathtt{x})}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 0} \qquad \dfrac{\dfrac{\dfrac{\Gamma(\mathtt{y\_s}) = \mathsf{Secret}}{\Gamma \vdash \mathtt{y\_s} : \mathsf{Secret}} \qquad \Gamma \vdash 1 : \mathsf{Public}}{\Gamma \vdash \mathtt{y\_s} - 1 : \mathsf{Secret}} \qquad \mathsf{Secret} \sqsubseteq \Gamma(\mathtt{y\_s})}{\dfrac{\dfrac{\Gamma(\mathtt{y\_s}) = \mathsf{Secret}}{\Gamma \vdash \mathtt{y\_s} : \mathsf{Secret}} \qquad \Gamma, \mathsf{Secret} \vdash \mathtt{y\_s} := \mathtt{y\_s} - 1}{\Gamma, \mathsf{Public} \vdash \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1} \qquad \dfrac{\Gamma \vdash 1 : \mathsf{Public} \qquad \mathsf{Public} \sqsubseteq \Gamma(\mathtt{x})}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 1}}{\Gamma, \mathsf{Public} \vdash \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1;\ \mathtt{x\_p} := 1}}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 0;\ \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1;\ \mathtt{x\_p} := 1}$$

FIGURE 5. Example typing derivation tree

Reformulation of memory agreement on public variables. Now that we have introduced a security environment Γ, we define memory agreement on public variables according to security environment Γ as follows

$$m_1 \sim_\Gamma m_2 \triangleq \forall x \,.\, \Gamma(x) = \mathsf{Public} \implies m_1(x) = m_2(x)$$

Whenever Γ is implicit (because it does not changes in our setting), we omit it for clarity, and simply write $m_1 \sim m_2$ (this overrides our earlier notation, but hopefully does not introduce too much confusion).

Exercise 7. Revisit example programs from Section 2.2 and examine whether these programs can be typed according to the typing rules of this section. The expectation is that all insecure programs are rejected by the type system (we postpone a formal proof of this), while there may be programs that despite being accepted by Definition 2, will be rejected by the typing rules; the latter illustrates imprecision of our enforcement.

Exercise 8. How can one change rule T-WHILE to satisfy termination-sensitive version of NI? While there are many ways to approach this, think of two possible techniques

(1) Modifying only typing of rule T-WHILE.

(2) Modifying the form of the typing judgment to include a *termination taint label*. This in general may increase permissiveness of the typing.

Remarks. The type system is based on [Volpano et al.(1996)Volpano, Smith, and Irvine]. The idea of behind such an analysis goes back to [Denning and Denning(1977)]; the literature frequently refers to this as "Denning-style" analysis.

4. Proof of soundness

This section establishes the formal relationship between the type system of Section 3 and the termination-insensitive noninterference given by Definition 2.[3]

Our main theorem is formulated as follows.

Theorem 1 (Soundness of the security type system)**.** *Given a program c such that $\Gamma, pc \vdash c$ then c satisfies Definition 2.*

To prove this theorem we introduce a number of auxiliary definitions.

4.1. Well-formedness and preservation.

Definition 3 (Well-formedness of memory w.r.t. a typing environment)**.** Given a memory m and a typing environment Γ say that m is *well-formed w.r.t.* Γ if $\mathsf{dom}(m) \subseteq \mathsf{dom}(\Gamma)$

Because the language is so small (the types only range over integers), the only requirement on the well-formedness is that if a variable appears in the program text it must be defined in the environment.

In the following, whenever we write $m_1 \sim_\Gamma m_2$ we also assume that m_1 and m_2 are well-formed w.r.t. Γ.

Well-formedness is lifted to configurations as follows.

Definition 4 (Well-formedness of configurations)**.** We say that a configuration $\langle c, m\rangle$ is well-formed w.r.t. a typing environment Γ and a level pc when both of the following hold:

(1) either c is stop, or the program is well-typed, i.e., $\Gamma, pc \vdash c$

(2) and, m is well-formed w.r.t. Γ.

[3] A mechanization of this proof in Coq is available at `https://github.com/aslanix/SmallStepNI`

$$\text{S-Assign-Pub}\quad \frac{\langle e, m\rangle \Downarrow v \qquad \Gamma(x) = \mathsf{Public}}{\langle x := e, m\rangle \twoheadrightarrow_{(x,v)} \langle \mathsf{stop}, m[x \mapsto v]\rangle} \qquad \text{S-Assign-Sec}\quad \frac{\langle e, m\rangle \Downarrow v \qquad \neg(\Gamma(x) = \mathsf{Public})}{\langle x := e, m\rangle \twoheadrightarrow_{\epsilon} \langle \mathsf{stop}, m[x \mapsto v]\rangle}$$

$$\text{S-Seq1-Ev}\quad \frac{\langle c_1, m\rangle \twoheadrightarrow_{\alpha} \langle \mathsf{stop}, m'\rangle}{\langle c_1; c_2, m\rangle \twoheadrightarrow_{\alpha} \langle c_2, m'\rangle} \qquad \text{S-Seq2-Ev}\quad \frac{\langle c_1, m\rangle \twoheadrightarrow_{\alpha} \langle c_1', m'\rangle \qquad c_1' \neq \mathsf{stop}}{\langle c_1; c_2, m\rangle \twoheadrightarrow_{\alpha} \langle c_1; c_2, m'\rangle}$$

FIGURE 6. Auxiliary semantics with events – selected rules.

Lemma 1 (Preservation of well-formedness)**.** *Assume we are given a typing environment* Γ*, level pc, and a configuration* $\langle c, m\rangle$*, such that the configuration is well-formed w.r.t.* Γ *and pc. Suppose this configurations takes a step*

$$\langle c, m\rangle \to \langle c', m'\rangle.$$

Then the resulting configuration $\langle c', m'\rangle$ *is also well-formed w.r.t.* Γ *and pc.*

Proof. By induction on c. □

4.2. **Auxiliary semantics and bridge relation.**

4.2.1. *Auxiliary semantics with events.* We define an auxiliary semantics that is operationally equivalent to the semantics in Section 1, but records additional information that is needed for the proof. We call the additional information recorded in the semantic transitions *public events*. In our setting, the only source of public events is public assignments – we record this by setting the form of public events to be (x, v), where x is the name of the variable that is being assigned to, and v is the value that is assigned to the variable. We abbreviate a single event by α.

$$\alpha ::= \epsilon \mid (x, v)$$

Figure 6 presents the selected rules for the auxiliary semantics. The omitted cases are trivial, and all produce an empty event ϵ.

As a sanity check, we establish the following technical lemma.

Lemma 2 (Adequacy of the semantics with events)**.** *Suppose we are given a typing environment* Γ*, a program c and memory m. Then* $\langle c, m\rangle \to \langle c', m'\rangle$ *if and only if there is an event* α *such that* $\langle c, m\rangle \twoheadrightarrow_{\alpha} \langle c', m'\rangle$*.*

Proof. We inspect each of the directions separately.

From standard to auxiliary: By induction of the steps relation $\to$. For the base cases, the only interesting case is assignment. In this case, we proceed by analysis of $\Gamma(x)$. We have the following cases

(1) Γ is defined on x and $\Gamma(x) = \mathsf{Public}$. In that case, the S-Assign-Pub is applied to pick the corresponding auxiliary transition.

(2) Γ is undefined on x or $\Gamma(x) = \mathsf{Secret}$. In that case, the rule S-Assign-Sec is applied to pick the corresponding auxiliary transition.

The inductive cases are straightforward.

From auxiliary to standard: By induction on $\twoheadrightarrow$; all cases are trivial.

□

Using Lemma 2 we can lift our proof of preservation to the auxiliary semantics.

$$\text{Bridge-Stop}\quad \frac{\langle c,m\rangle \twoheadrightarrow_\epsilon \langle \mathsf{stop},m'\rangle}{\langle c,m\rangle \curvearrowright_\epsilon^0 \langle \mathsf{stop},m'\rangle} \qquad \text{Bridge-Public}\quad \frac{\langle c,m\rangle \twoheadrightarrow_\alpha \langle c',m'\rangle \qquad \alpha\neq\epsilon}{\langle c,m\rangle \curvearrowright_\alpha^0 \langle c',m'\rangle}$$

$$\text{Bridge-Multi}\quad \frac{\langle c,m\rangle \twoheadrightarrow_\epsilon \langle c',m'\rangle \qquad c'\neq \mathsf{stop} \qquad \langle c',m'\rangle \curvearrowright_\alpha^n \langle c'',m''\rangle}{\langle c,m\rangle \curvearrowright_\alpha^{n+1} \langle c'',m''\rangle}$$

FIGURE 7. Bridge relation

Lemma 3 (Preservation of typing for the auxiliary semantics)**.** *Assume we are given a typing environment* Γ*, level pc, and a configuration* $\langle c,m\rangle$*, such that the configuration is well-formed w.r.t.* Γ *and pc. Suppose this configurations takes a step*

$$\langle c,m\rangle \twoheadrightarrow \alpha\langle c',m'\rangle.$$

Then the resulting configuration $\langle c',m'\rangle$ *is also well-formed w.r.t.* Γ *and pc.*

Proof. Immediate from Lemma 1 and Lemma 2. □

4.2.2. *Bridge relation.* Using the auxiliary semantics, we introduce our main techincal vehicle of this proof – a so-called *bridge relation*. We say that configuration $\langle c,m\rangle$ *bridges to* configuration $\langle c',m'\rangle$ when $\langle c',m'\rangle$ is the first configuration reachable from $\langle c,m\rangle$ with a public assignment event, or a $\langle c',m'\rangle$ is a terminal configuration. Bridge relation is defined in terms of the auxiliary semantics with events. Bridge relations are indexed by the number of the intermediate steps leading to the event; this is needed to apply the strong induction principle in the proof of noninterference for the bridge relation.

We denote the bridge relation as $\langle c,m\rangle \curvearrowright_\alpha^n \langle c',m'\rangle$, where α is the event produced by the transition to the configuration $\langle c',m'\rangle$, and n is the number of the intermediate steps. Figure 7 presents this relation. Note that this relation is not reflexive.

4.2.3. *Properties of bridge relation.* We observe that in order for a sequential composition of two commands to produce an event, it must be that either the first command is consumed silently (emitting an empty event ϵ), or it is the first command that actually produces the event. This is formalized by the following lemma.

Lemma 4 (Bridge of sequential composition)**.** *Given a typing environment* Γ*, a sequential composition of two commands* c_1 *and* c_2 *such that* $\langle c_1;c_2,m\rangle \curvearrowright_\alpha^n \langle c',m'\rangle$ *then one of the following holds*

(1) *it must be that* $n>0$ *and there are* k *and* m_1' *such that* $k<n$ *and* $\langle c_1,m\rangle \curvearrowright_\epsilon^k \langle \mathsf{stop},m_1'\rangle$ *and* $\langle c_2,m_1'\rangle \curvearrowright_\alpha^{n-k-1} \langle c',m'\rangle$

(2) *or,* $\alpha\neq\epsilon$ *and there is* c_1' *such that* $\langle c_1,m\rangle \curvearrowright_\alpha^n \langle c_1',m'\rangle$ *and* $c' = \begin{cases} c_1';c_2 & \text{if } c_1'\neq \mathsf{stop} \\ c_2 & o/w \end{cases}$.

Proof. By inspection of the rules in the bridge relation and the associated rules of the auxiliary semantics. □

4.3. **Noninterference for expressions.** Our proof of noninterference for commands relies on noninterference for expressions that we formalize via the following lemma below.

Lemma 5 (Noninterference for expressions)**.** *Given a typing environment* Γ *and two memories* m_1 *and* m_2 *such that* $m_1 \sim_\Gamma m_2$*, and an expression* e *such that* $\Gamma \vdash e : \ell$*, such that* $\langle e, m\rangle \Downarrow v_1$ *and* $\langle e, m\rangle \Downarrow v_2$*, then we have that*

$$\ell \sqsubseteq \mathsf{Public} \implies v_1 = v_2.$$

Proof. By induction on the typing derivation $\Gamma \vdash e : \ell$. □

4.4. **Properties of public-memory–equivalence.** We establish a number of useful technical lemmas for public-memory–equivalence.

Lemma 6 (Transitivity of $\sim_\Gamma$)**.** *Given* Γ *and* m_1, m_2, m_3*, if* $m_1 \sim_\Gamma m_2$ *and* $m_2 \sim_\Gamma m_3$ *then* $m_1 \sim_\Gamma m_3$*.*

Proof. Immediate by definition of $\sim_\Gamma$. □

Lemma 7 (Public updates preserve pubilc-memory–equivalence)**.** *Given* Γ *and two memories* m_1 *and* m_2 *such that* $m_1 \sim_\Gamma m_2$ *and a variable* x *such that* $\Gamma(x) = \mathsf{Public}$*. Then for all values* v*, it must be that* $m_1[x \mapsto v] \sim_\Gamma m_2[x \mapsto v]$*.*

Proof. Immediate by definition of memory update and definition of $\sim_\Gamma$. □

4.5. **High steps.** An important property of our type system is that it disallows assignments to public variables when the program counter level is secret. In particular, this means that programs that are well-typed when $pc = \mathsf{Secret}$ it must hold that the initial and the final memories are public-memory–equivalent.

The following lemma expresses the above intuition for a single step, allowing a straightforward generalization to a number of steps.

Lemma 8 (Commands typed in secret context are not making public assignments)**.** *Given a typing environment* Γ*, and program* c *such that* $\Gamma, \mathsf{Secret} \vdash c$*, memory* m*, such that* $\langle c, m\rangle \rightarrow_\alpha \langle c', m'\rangle$ *then* $\alpha = \epsilon$ *and* $m \sim_\Gamma m'$*.*

Proof. By induction on the structure of c. □

4.6. **Noninterference for bridge relation.** We are now ready to formulate our noninterference for the bridge relation.

Theorem 2 (Noninterference for bridge)**.** *Suppose we are given a typing environment* Γ*, two memories* m_1 *and* m_2 *such that* $m_1 \sim_\Gamma m_2$*, and program* c *such that* $\Gamma, pc \vdash c$*. Suppose further that we have a pair of (potentially empty) events* α_1, α_2*, some* n *and* n_2*, and two configurations* $\langle c_1', m_1'\rangle$ *and* $\langle c_2', m_2'\rangle$ *such that*

$$\langle c, m_1\rangle \curvearrowright^n_{\alpha_1} \langle c_1', m_1'\rangle \qquad \text{and} \qquad \langle c, m_2\rangle \curvearrowright^{n_2}_{\alpha_2} \langle c_2', m_2'\rangle$$

Then, it must be that

1. *the final commands are the same:* $c_1' = c_2'$*.*
2. *the final memories agree on public variables:* $m_1' \sim_\Gamma m_2'$*.*
3. $\alpha_1 \neq \epsilon$ *if and only if* $\alpha_2 \neq \epsilon$*.*
4. *if* $\alpha_1 \neq \epsilon$ *then* $\alpha_1 = \alpha_2$*.*

Proof. The proof is "two-dimensional" – it uses two nested inductions: an outer induction is on the number of steps n, with the inner inductions on the structure of the commands. We examine this in detail.

Outer base case $n = 0$**:** We proceed by an inner induction on the structure of c.

Case c **is** skip**:** For both runs, the only applicable rule is Bridge-Stop that produces stop-configurations. Moreover, the memories do not change, and both events α_1 and α_2 must be ϵ.

Case c **is** $x := e$**:** We examine the bridge relation $\langle x := e, m\rangle \curvearrowright_{\alpha}^{0} \langle c_1', m_1'\rangle$. The only two possible bridge rules are Bridge-Stop and Bridge-Public (rule Bridge-Multi is not applicable because $n = 0$). We examine each of the cases.

Case Bridge-Stop**:** It must be that $\langle x := e, m\rangle \twoheadrightarrow_{\epsilon} \langle c_1', m_1'\rangle$. This is only possible by rule S-Assign-Sec, in particular we have that $\neg(\Gamma(x) = \mathsf{Public})$. By Lemma 3, it must be that Γ is defined for x and hence it must be that $\Gamma(x) = \mathsf{Secret}$. This means that assignment to variable x does not change public-memory–equivalence, and therefore it must be that $m_1 \sim_\Gamma m_1'$.

Now that we have established that $\Gamma(x) = \mathsf{Secret}$, we observe that the bridge transition for the second run must also be produced by an assignment to secret variable x (also via Bridge-Stop and S-Assign-Sec). Thus, $m_2 \sim_\Gamma m_2'$.

We are then done by (two applications of) transitivity of $\sim_\Gamma$ (Lemma 6), observing that the final commands are stop.

Case Bridge-Public**:** The only applicable rule in the auxiliary semantics is S-Assign-Pub. This means that $\Gamma(x) = \mathsf{Public}$.

Now that we have established that $\Gamma(x) = \mathsf{Public}$, we observe that the bridge transition for the second run must also be produced by an assignment to public variable x (also via Bridge-Public and S-Assign-Pub).

Because our program is well-typed, it must be that $\Gamma, pc \vdash x := e$. The only applicable typing rule is T-Assign. This rule requires that it must be that $pc \sqcup \ell \sqsubseteq \mathsf{Public}$, where $\Gamma \vdash e : \ell$. This means, in particular, that it must be that $\ell \sqsubseteq \mathsf{Public}$. By Lemma 5, it must be that $v_1 = v_2$, where v_1 and v_2 are the results of evaluating expression e in two respective runs, i.e., $\langle e, m_1\rangle \Downarrow v_1$ and $\langle e, m_2\rangle \Downarrow v_2$.

Then we are done by Lemma 7, observing that the resulting commands must be stop, and that for the produced events it holds that $\alpha_1 = (x, v_1) = (x, v_2) = \alpha_2$.

Case c **is** $d_1; d_2$**:** We apply Lemma 4, and observe that only one of the cases indicated by that lemma is possible (the other one would require $0 > 0$). So, it must be that $\alpha_1 \neq \epsilon$ and there is d_1' such that

$$\langle d_1, m_1\rangle \curvearrowright_{\alpha_1}^{0} \langle d_1', m_1'\rangle. \tag{4.1}$$

and $c_1' = \begin{cases} d_1'; d_2 & \text{if } d_1' \neq \mathsf{stop} \\ d_2 & \text{o/w} \end{cases}$

Given $\langle d_1; d_2, m_2\rangle \curvearrowright_{\alpha_2}^{n_2} \langle c_2', m_2'\rangle$, then by Lemma 4, we have two possibilities:

(1) it must be that $n_2 > 0$ and there are k and m_2'' such that $k < n_2$ and

$$\langle d_1, m_2\rangle \curvearrowright_{\epsilon}^{k} \langle \mathsf{stop}, m_2''\rangle \tag{4.2}$$

and $\langle d_2, m_2''\rangle \curvearrowright_{\alpha_2}^{n_2-k-1} \langle c_2', m_2'\rangle$.

Using equations 4.1 and 4.2 we can apply the inner induction hypothesis for d_1 to arrive at a contradiction that $\alpha_1 = \epsilon$. Thus, this case is impossible.

(2) or, $\alpha_2 \neq \epsilon$, and there is d_1'' s.t. $\langle d_1, m_2\rangle \curvearrowright_{\alpha_2}^{n_2} \langle d_1'', m_2'\rangle$ and $c_2' = \begin{cases} d_1''; d_2 & \text{if } d_1'' \neq \mathsf{stop} \\ d_2 & \text{o/w} \end{cases}$.

In this case, we apply the induction hypothesis to d_1 to conclude that

- $d_1' = d_1''$.
- $m_1' \sim_\Gamma m_2'$
- $\alpha_1 \neq \epsilon$ if and only if $\alpha_2 \neq \epsilon$.
- if $\alpha_1 \neq \epsilon$ then $\alpha_1 = \alpha_2$.

Using this, we obtain that it must be that $c_1' = c_2'$, which together with the above facts closes this subcase.

Case c **is** $\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$**:** Not applicable, because $n = 0$.

Case c **is** $\mathsf{while}\ e\ \mathsf{do}\ c$**:** Not applicable, because $n = 0$.

Outer inductive case: We proceed by an inner induction on the structure of c.

Case c **is** skip**:** Not applicable, because $n > 0$.

Case c **is** $x := e$**:** Not applicable, because $n > 0$.

Case c **is** $d_1; d_2$**:** We apply Lemma 4 to each of the runs to obtain the following.

(1) For the first run, we have the two possibilities

(a) it must be that $n_1 > 0$ and there are k_1 and m_1'' such that $k_1 < n$ and

$$\langle d_1, m_1\rangle \curvearrowright_\epsilon^{k_1} \langle \mathsf{stop}, m_1''\rangle \tag{4.3}$$

and

$$\langle d_2, m_1''\rangle \curvearrowright_{\alpha_1}^{n-k_1-1} \langle c_1', m_1'\rangle \tag{4.4}$$

(b) or, $\alpha_1 \neq \epsilon$, and there is d_1'' such that

$$\langle d_1, m_1\rangle \curvearrowright_{\alpha_1}^{n} \langle d_1'', m_1'\rangle \tag{4.5}$$

and $c_1' = \begin{cases} d_1''; d_2 & \text{if } d_1'' \neq \mathsf{stop} \\ d_2 & \text{o/w} \end{cases}$.

(2) Similarly, for the second run, we have the two possibilities

(a) it must be that $n_2 > 0$ and there are k_2 and m_2'' such that $k_2 < n_2$ and

$$\langle d_1, m_2\rangle \curvearrowright_\epsilon^{k_2} \langle \mathsf{stop}, m_2''\rangle \tag{4.6}$$

and

$$\langle d_2, m_2''\rangle \curvearrowright_{\alpha_2}^{n_2-k_2-1} \langle c_2', m_2'\rangle \tag{4.7}$$

(b) or, $\alpha_2 \neq \epsilon$, and there is d_2'' such that

$$\langle d_1, m_2\rangle \curvearrowright_{\alpha_2}^{n_2} \langle d_2'', m_2'\rangle \tag{4.8}$$

and $c_2' = \begin{cases} d_1''; d_2 & \text{if } d_2'' \neq \mathsf{stop} \\ d_2 & \text{o/w} \end{cases}$.

This gives four cases in total to consider: (1a,2a), (1a,2b), (1b, 2a), and (1b, 2b). Out of these, the two – namely (1a,2b) and (1b,2a) – are impossible. This can be seen by applying the inner induction hypothesis to d_1 to arrive at contradicting conclusions. In case of (1a, 2b), the contradicting conclusion is $\alpha_2 = \epsilon$, and in case of (1b,2a), it is that $\alpha_1 = \epsilon$. We consider the remaining two cases:

Case (1a,2a): We apply the inner induction hypothesis twice. First, we apply it to d_1 which gives us that $m_1'' \sim_\Gamma m_2''$. From here, we are done by applying the induction hypothesis to d_2.

Case (1b,2b): Here, we are done immediately by applying the inner induction hypothesis to d_1.

Case c **is** $\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$**:** By T-If, it must be that $\Gamma, pc \sqcup \ell \vdash c_i$, for $i = 1, 2$, where $\Gamma \vdash e : \ell$. We consider two cases

Case $pc \sqcup \ell \sqsubseteq \mathsf{Public}$**:** . In this case, it must be that by Lemma 5, both runs take the same branch, and we are done by the inner induction hypothesis.

Case $pc \sqcup \ell \not\sqsubseteq \mathsf{Public}$**:** In this case, by Lemma 8, it must be that neither of the runs update public variables, and we are done by repeated applications of Lemma 6.

Case c **is** $\mathsf{while}\ e\ \mathsf{do}\ c$**:** Immediately by the outer induction hypothesis.

□

4.7. **Bridge adequacy.**

Lemma 9 (Bridge adequacy)**.** *Given a program c such that $\Gamma, pc \vdash c$, and memory m that is well-formed w.r.t. Γ, and such that $\langle c, m\rangle \to^n \langle \mathsf{stop}, m_{final}\rangle$, and some m_{final}, then there are c', m', α, k, and n' such that $\langle c, m\rangle \curvearrowright^k_\alpha \langle c', m'\rangle$ and $\langle c', m'\rangle \to^{n'} \langle \mathsf{stop}, m_{final}\rangle$, where $k + n' + 1 = n$.*

Proof. By induction on n. Base case $n = 0$ is trivial. For the inductive case, we assume that the lemma holds for $n - 1$ steps, and consider the case of n steps.

We have that

$$\langle c, m\rangle \to \langle c', m'\rangle \qquad \text{and} \qquad \langle c', m'\rangle \to^{n-1} \langle \mathsf{stop}, m_{final}\rangle$$

By Lemma 2, there is α such that $\langle c, m\rangle \twoheadrightarrow_\alpha \langle c', m'\rangle$.

We consider two possibilities:

Case $\alpha \neq \epsilon$**:** Then we are done by rule Bridge-Public, and we set $k = 0$ and $n' = n - 1$.

Case $\alpha = \epsilon$**:** We have two cases

(1) $n - 1 = 0$. In this case we are done by Bridge-Stop, and once again $k = 0$ and $n' = n - 1$.

(2) $n - 1 > 0$. In this case, it must be that $c' \neq \mathsf{stop}$. By the induction hypothesis $\langle c', m'\rangle \curvearrowright^{k''}_\alpha \langle c'', m''\rangle$, and $\langle c'', m''\rangle \to^{n''} \langle \mathsf{stop}, m_{final}\rangle$, where $k'' + n'' + 1 = n - 1$. Then, using Bridge-Multi we have that $\langle c, m\rangle \curvearrowright^{k''+1}_\alpha \langle c'', m''\rangle$, so we set $k = k'' + 1$ and $n' = n''$.

□

4.8. **Revisiting Theorem 1.** We are now ready to combine all the pieces necessary for the proof of Theorem 1.

Restatement of Theorem 1 (Soundness of the security type system). Given a program c such that $\Gamma, pc \vdash c$ then c satisfies Definition 2.

Proof. Unfolding Definition 2 (and instantiating it to equivalence under Γ), we need to show that for all pairs of memories m_1 and m_2 such that $m_1 \sim_\Gamma m_2$, and

$$\langle c, m_1\rangle \to^* \langle \mathsf{stop}, m'_1\rangle \tag{4.9}$$

and

$$\langle c, m_2\rangle \to^* \langle \mathsf{stop}, m'_2\rangle \tag{4.10}$$

it holds that $m'_1 \sim_\Gamma m'_2$.

From equations 4.9 and 4.10, there are n_1 and n_2 such that

$$\langle c, m_1\rangle \to^{n_1} \langle \mathsf{stop}, m_1'\rangle \qquad \text{and} \qquad \langle c, m_2\rangle \to^{n_2} \langle \mathsf{stop}, m_2'\rangle$$

We proceed by strong induction on n_1.

Base case $n_1 = 0$**:** In this case, c must be stop, and we are done immediately.

Inductive case: In this case, c is not stop. By Lemma 9, applied to both runs, we have:

(1) for the first run: $\langle c, m_1\rangle \curvearrowright^{k_1}_{\alpha_1} \langle c_1', m_1''\rangle$ and $\langle c_1', m_1''\rangle \to^{n_1'} \langle \mathsf{stop}, m_1'\rangle$, with $n_1' < n_1$.

(2) for the second run: $\langle c, m_2\rangle \curvearrowright^{k_2}_{\alpha_2} \langle c_2', m_2''\rangle$ and $\langle c_2', m_2''\rangle \to^{n_2'} \langle \mathsf{stop}, m_2'\rangle$.

By Theorem 2, we get that $c_1' = c_2'$ and $m_1'' \sim_\Gamma m_2''$. By preservation, it must be that either $c_1' = \mathsf{stop}$ or $\Gamma, pc \vdash c_1'$. In case $c_1' = \mathsf{stop}$,we are done immediately, and it must be that $m_i' = m_i'', i = 1, 2$. Otherwise, we can apply the induction hypothesis for n_1' to obtain the public-memory–equivalence of the final memories, as desired.

□

References

[Denning and Denning(1977)] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.

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

[Plotkin(1981)] G. D. Plotkin. A structural approach to operational semantics. 1981.

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

[Volpano et al.(1996)Volpano, Smith, and Irvine] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[Winskel(1993)] G. Winskel. *The Formal Semantics of Programming Languages*. 3rd edition, 1993.