Dynamic Noninterference

by Casper Rysgaard

Introduction

These notes cover the basics of making a dynamic noninterference language. In Askarov's original notes, he proves this for a statically checked language. This proof is based upon the same idea he uses there. It can be found here: 'github.com/aslanix/SmallStepNI/blob/master/infoflow-basics.pdf'.

1 Small Imperative Language

We consider a simple imperative language with the following grammar:

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

Here n is any number, which for simplicity is only natural numbers in this proof, x is a string variable, and `op` ranges over any operator; in this proof we only consider the addition operator.

We also consider for simplicity two levels of security; `Public` and `Secret`. This is based on the intuition for noninterference, that can be found in the next section.

We use big-step evaluation for expressions denoted as $\langle e, m\rangle \Downarrow \langle v, l\rangle$, where e is the expression to evaluate, m is the memory it is evaluated in, v is the value of the evaluation, and l is the level of the expression. The semantics can be seen in Figure 1.

E-Const
$$\frac{}{\langle n, m\rangle \Downarrow \langle n, \texttt{Public}\rangle}$$

E-Var
$$\frac{m(x) = v \qquad \texttt{levelof}(x) = l}{\langle x, m\rangle \Downarrow \langle v, l\rangle}$$

E-Op
$$\frac{\langle e_i, m\rangle \Downarrow \langle v_i, l_i\rangle,\ i = 1, 2 \qquad v = v_1\ \texttt{op}\ v_2 \qquad l' = \begin{cases} \texttt{Public} & \text{if } l_1 = l_2 = \texttt{Public} \\ \texttt{Secret} & \text{otherwise} \end{cases}}{\langle e_1\ \texttt{op}\ e_2, m\rangle \Downarrow \langle v, l'\rangle}$$

Figure 1: Semantics for expressions

Memories are seen as complete functions from a string x to a value v. Here $m[x \mapsto v]$ is shorthand for memory update:

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

Here we define the $\texttt{levelof}(x)$ to be a look up, that finds the level of the string x. The level of a string never chances during execution.

In the Coq proof, the memory also takes care of finding the level of a string, and thus it is a function to a tuple containing the value and the level.

For commands we also introduce the auxiliary command `stop`, which is used to determine terminating configurations. Semantics for commands is given as $\langle c, m, \vec{l}\rangle \rightarrow \langle c', m', \vec{l'}\rangle$, where c is the starting command, m is the starting memory, $\vec{l}$ is the starting stack of levels, c' is the updated command or `stop`,

m' is the updated memory, and $\vec{l'}$ is the updated stack of levels. We refer to $\langle c,\, m,\, \vec{l}\rangle$ as a semantic configuration.

Rules for the semantic transitions are given in Figure 2, where $l :: \vec{l}$ denotes a stack consisting of first l, then followed by the rest of the stack $\vec{l}$.

$$\text{S-Skip}\quad \frac{}{\langle \texttt{skip},\, m,\, \vec{l}\rangle \rightarrow \langle \texttt{stop},\, m,\, \vec{l}\rangle}$$

$$\text{S-Assign}\quad \frac{\langle e,\, m\rangle \Downarrow \langle v,\, l\rangle \qquad \texttt{levelof}(x) = \texttt{Public} \Longrightarrow \forall\, l' \in \vec{l}\,:\, l' = \texttt{Public}}{\langle x := e,\, m,\, \vec{l}\rangle \rightarrow \langle \texttt{stop},\, m[x \mapsto v], \vec{l}\rangle}$$

$$\text{S-Seq1}\quad \frac{\langle c_1,\, m,\, \vec{l}\rangle \rightarrow \langle \texttt{stop},\, m',\, \vec{l'}\rangle}{\langle c_1\,;\, c_2,\, m,\, \vec{l}\rangle \rightarrow \langle c_2,\, m',\, \vec{l'}\rangle} \qquad \text{S-Seq2}\quad \frac{\langle c_1,\, m,\, \vec{l}\rangle \rightarrow \langle c_1',\, m',\, \vec{l'}\rangle \qquad c_1' \neq \texttt{stop}}{\langle c_1\,;\, c_2,\, m,\, \vec{l}\rangle \rightarrow \langle c_1'\,;\, c_2,\, m',\, \vec{l'}\rangle}$$

$$\text{S-If1}\quad \frac{\langle e,\, m\rangle \Downarrow \langle v,\, l\rangle \qquad v \neq 0}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\, m,\, \vec{l}\rangle \rightarrow \langle c_1\,;\, \texttt{join},\, m,\, l :: \vec{l}\rangle}$$

$$\text{S-If2}\quad \frac{\langle e,\, m\rangle \Downarrow \langle v,\, l\rangle \qquad v = 0}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\, m,\, \vec{l}\rangle \rightarrow \langle c_2\,;\, \texttt{join},\, m,\, l :: \vec{l}\rangle}$$

$$\text{S-While}\quad \frac{}{\langle \texttt{while}\ e\ \texttt{do}\ c,\, m,\, \vec{l}\rangle \rightarrow \langle \texttt{if}\ e\ \texttt{then}\ c\,;\, \texttt{while}\ e\ \texttt{do}\ c\ \texttt{else}\ \texttt{skip},\, m,\, \vec{l}\rangle}$$

$$\text{S-Join}\quad \frac{}{\langle \texttt{join},\, m,\, l :: \vec{l}\rangle \rightarrow \langle \texttt{stop},\, m,\, \vec{l}\rangle}$$

Figure 2: Semantics for commands

We write $\langle c,\, m,\, \vec{l}\rangle \rightarrow^* \langle c',\, m',\, \vec{l'}\rangle$ when configuration $\langle c,\, m,\, \vec{l}\rangle$ can reach configuration $\langle c',\, m',\, \vec{l'}\rangle$ in zero or more steps.

The idea behind the `join` is to tell the command, that the `if` that was started, has ended, so that it is no longer in the context of it.

2 Noninterference

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

$$m_1 \sim m_2 \triangleq \forall\, x\ :\ \texttt{levelof}(x) = \texttt{Public} \Rightarrow m_1(x) = m_2(x)$$

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

$$\langle c,\, m_1,\, \vec{l}\rangle \rightarrow^* \langle \texttt{stop},\, m_1',\, \vec{l'}\rangle$$

and

$$\langle c,\, m_2,\, \vec{l}\rangle \rightarrow^* \langle \texttt{stop},\, m_2',\, \vec{l'}\rangle$$

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

3 Proof of Soundness

This section establishes the formal relationship between the termination-insensitive noninterference given by Definition 2 and the dynamically checked semantics for commands given in Figure 2.

Our main theorem is formulated as follows.

Theorem 1 (Soundness of the dynamic semantics). *Given a program c such that* $\langle c,\, m,\, \vec{l}\rangle \rightarrow^* \langle c',\, m',\, \vec{l'}\rangle$ *then c satisfies Definition 2.*

To prove this we introduce a number of auxiliary definitions and lemmas.

3.1 Well-formedness and preservation

Definition 3 (Well-formedness of a command w.r.t. a number). Given a command c and a number n, we say that c is well-formed if either c is `stop` and $n = 0$ or $\textsc{wf}(c,\, n)$ according to the rules defined in Figure 3.

$$\frac{}{\textsc{wf}(\text{Join},\ 1)}\ \textsc{WF-Join} \qquad \frac{c \neq c_1; c_2 \qquad c \text{ is } \texttt{stop} \text{ free} \qquad c \text{ is } \texttt{join} \text{ free}}{\textsc{wf}(c,\ 0)}\ \textsc{WF-JoinFree}$$

$$\frac{\textsc{wf}(c_1,\ k_1) \qquad \textsc{wf}(c_2,\ k_2) \qquad k_1, k_2 \geq 0}{\textsc{wf}(c_1; c_2,\ k_1 + k_2)}\ \textsc{WF-Seq}$$

Figure 3: Rules for Well-Formedness of Non-`stop` Commands

The well-formedness is thus a way to count the number of `join`'s in a command, as well as making sure that it does not contain any `stop`'s and that all `join`'s are not in the branch of an `if`. This is also why $k_1, k_2 \geq 0$ in the WF-Seq rule, as a command cannot have a negative amount of `join`'s.

Lemma 1 (`join` free well formedness). *Given a command c that is both* `stop` *and* `join` *free, then c is well formed w.r.t. 0.*

Proof. By induction on c.

Lemma 2 (Well formed deterministics). *Given a command c and two numbers n and m, such that c is well formed w.r.t. both n and m, then* $n = m$.

Proof. By induction on c.

Lemma 3 (Preservation of well formedness). *Given a command c, a stack* $\vec{l}$ *and a number n, such that c is well formed w.r.t.* $|\vec{l}| - n$, *and* $\langle c,\, m,\, \vec{l}\rangle \rightarrow \langle c',\, m',\, \vec{l'}\rangle$, *then the resulting command* c' *is well formed w.r.t.* $|\vec{l'}| - n$.

Proof. By induction on c. Here Lemma 1 and 2 can be combined to find the number n in some of the cases. The number is added, as in the case of sequential, we can apply the induction hypothesis.

3.2 Auxiliary semantics

We define a set of auxiliary semantics that operationally is the same as those that can be seen in Figure 2, but which additionally records information that is needed for the proof. Specifically we record all public events. In our setting the only public events are the public assignments. We record this as a tuple

(x, v), where x is the name of the public variable that was assigned to, and v is the value assigned to it. This thus gives us the following grammar for events α.

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

Some selected auxiliary semantics can be seen in Figure 4. Here the omitted cases are trivial, and all produce an empty event ϵ.

$$\text{S-Assign-Pub}\quad \frac{\langle e, m\rangle \Downarrow \langle v, l\rangle \qquad \texttt{levelof}(x) = \texttt{Public} \qquad \forall l' \in \vec{l}\colon\ l' = \texttt{Public}}{\langle x := e, m, \vec{l}\rangle \twoheadrightarrow_{(x,v)} \langle \texttt{stop}, m[x \mapsto v], \vec{l}\rangle}$$

$$\text{S-Assign-Sec}\quad \frac{\langle e, m\rangle \Downarrow \langle v, l\rangle \qquad \texttt{levelof}(x) \neq \texttt{Public}}{\langle x := e, m, \vec{l}\rangle \twoheadrightarrow_{\epsilon} \langle \texttt{stop}, m[x \mapsto v], \vec{l}\rangle}$$

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

Figure 4: Auxiliary semantics with events - selected rules

In order to convert between the two types of semantics, for the final proof, we establish the following conversion lemma.

Lemma 4 (Adequacy of the semantics with events). *Given a command c, a memory m and a stack $\vec{l}$, then $\langle c, m, \vec{l}\rangle \to \langle c', m', \vec{l'}\rangle$ if and only if there is an event α such that $\langle c, m, \vec{l}\rangle \twoheadrightarrow_{\alpha} \langle c', m', \vec{l'}\rangle$.*

Proof. We inspect each of the directions separately.

From standard to auxiliary: By induction on the step relation $\to$. In the case of an assignment, we look at all the cases for $\texttt{levelof}(x)$. The rest of the cases are trivial.

From auxiliary to standard: By induction on the auxiliary step relation $\twoheadrightarrow$. All cases are trivial.

Lemma 5 (Preservation of well formedness for event steps). *Given a command c, a stack $\vec{l}$ and a number n, such that c is well formed w.r.t. $|\vec{l}| - n$, and $\langle c, m, \vec{l}\rangle \twoheadrightarrow_{\alpha} \langle c', m', \vec{l'}\rangle$, then the resulting command c' is well formed w.r.t. $|\vec{l'}| - n$.*

Proof. Immediate from Lemma 3 and Lemma 4.

3.3 Bridge relation

Using the auxiliary semantics, we introduce the main engine behind the master proof - as so called bridge relation. The idea is to leave the command running until it produces a public event, as this is the only time the memories can start to not agree. We say that the configuration $\langle c, m, \vec{l}\rangle$ bridges to configuration $\langle c', m', \vec{l'}\rangle$, where $\langle c', m', \vec{l'}\rangle$ is the first configuration on the path that either produces a public event, or terminates. The bridge relation is also indexed by the number of intermediate steps, as this is needed to apply induction in the proofs.

We denote the bridge relation as $\langle c, m, \vec{l}\rangle \curvearrowright^{n}_{\alpha} \langle c', m', \vec{l'}\rangle$, where n is the number of intermediate steps, α is the event the last of the steps produces, before it reaches the configuration $\langle c', m', \vec{l'}\rangle$. The semantics of the relation can be seen in Figure 5.

$$\textsc{Bridge-Stop}\quad \frac{\langle c, m, \vec{l}\rangle \twoheadrightarrow_{\epsilon} \langle \texttt{stop}, m', \vec{l'}\rangle}{\langle c, m, \vec{l}\rangle \curvearrowright_{\epsilon}^{0} \langle \texttt{stop}, m', \vec{l'}\rangle} \qquad \textsc{Bridge-Public}\quad \frac{\langle c, m, \vec{l}\rangle \twoheadrightarrow_{(x,v)} \langle c', m', \vec{l'}\rangle}{\langle c, m, \vec{l}\rangle \curvearrowright_{(x,v)}^{0} \langle c', m', \vec{l'}\rangle}$$

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

Figure 5: Bridge relation

First we establish preservation of well formedness for the bridge relation.

Lemma 6 (Preservation of well formedness for bridge relation). *Given a command c, a stack $\vec{l}$ and a number n, such that c is well formed w.r.t. $|\vec{l}| - n$, and $\langle c, m, \vec{l}\rangle \curvearrowright_{\alpha}^{k} \langle c', m', \vec{l'}\rangle$, then the resulting command c' is well formed w.r.t. $|\vec{l'}| - n$.*

Proof. By induction on k.

We next observe that in order for a sequential command to produce an event, it must either be that the first command produces the event, or that it terminates silently, and the the second command produce the event. This is formalized in the following lemma.

Lemma 7 (Bridge of sequential compositions). *Given two commands c_1 and c_2, such that $\langle c_1; c_2, m, \vec{l}\rangle \curvearrowright_{\alpha}^{n} \langle c', m', \vec{l'}\rangle$ then one of the following two cases must hold*

(1) *$\alpha \neq \epsilon$ and there exists a c_1' such that $\langle c_1, m, \vec{l}\rangle \curvearrowright_{\alpha}^{n} \langle c_1', m', \vec{l'}\rangle$ and $c' = \begin{cases} c_1'; c_2 & \text{if } c_1' \neq \texttt{stop} \\ c_2 & \text{otherwise} \end{cases}$*

(2) *or $n > 0$ and there exists a k, m_1', and $\vec{l_1'}$, such that $k < n$ and $\langle c_1, m, \vec{l}\rangle \curvearrowright_{\epsilon}^{n} \langle \texttt{stop}, m_1', \vec{l_1'}\rangle$ and $\langle c_2, m_1', \vec{l_1'}\rangle \curvearrowright_{\alpha}^{n-k-1} \langle c', m', \vec{l'}\rangle$*

Proof. By induction on n.

3.4 Deterministic and noninterference for expressions

Lemma 8 (Deterministic level for expressions). *Given two memories m_1 and m_2, such that $m_1 \sim m_2$, and an expression e such that $\langle e, m_1\rangle \Downarrow \langle v_1, l_1\rangle$ and $\langle e, m_2\rangle \Downarrow \langle v_2, l_2\rangle$, then $l_1 = l_2$.*

Proof. By induction on e.

Lemma 9 (Noninterference for expressions). *Given two memories m_1 and m_2, such that $m_1 \sim m_2$, and an expression e such that $\langle e, m_1\rangle \Downarrow \langle v_1, \texttt{Public}\rangle$ and $\langle e, m_2\rangle \Downarrow \langle v_2, \texttt{Public}\rangle$, then $v_1 = v_2$.*

Proof. By induction on e.

3.5 Noninterference for bridge relation

Lemma 10 (`join` free bridge in a secret environment makes no public assignments). *Given a command c such that c is well formed w.r.t. 0, and a stack $\vec{l}$, such that there $\exists$ $\texttt{Secret} \in \vec{l}$, and $\langle c, m, \vec{l}\rangle \curvearrowright_{\alpha}^{n} \langle c', m', \vec{l'}\rangle$, then it must hold that $c' = \texttt{stop}$ and $m \sim m'$ and $\vec{l} = \vec{l'}$ and $\alpha = \epsilon$*

Proof. By a strong outer induction on n, with an inner induction on c.

We are now ready to formalize the noninterference property for the bridge relation.

Theorem 2 (Noninterference for bridge). *Given a command c, a stack $\vec{l}$ and a number k, such that c is well formed w.r.t. $|\vec{l}| - k$, and two memories m_1 and m_2, such that $m_1 \sim m_2$ and*

$$\langle c, m_1, \vec{l}\rangle \curvearrowright_{\alpha_1}^{n_1} \langle c_1', m_1', \vec{l_1'}\rangle \quad \text{and} \quad \langle c, m_2, \vec{l}\rangle \curvearrowright_{\alpha_2}^{n_2} \langle c_2', m_2', \vec{l_2'}\rangle$$

then it must be that $c_1' = c_2'$ and $m_1' \sim m_2'$ and $\vec{l_1'} = \vec{l_2'}$ and $\alpha_1 = \alpha_2$.

Proof. By a strong outer induction on n, with an inner induction on c.

3.6 Bridge adequacy

Lemma 11 (Bridge adequacy). *Given a command c, a stack $\vec{l}$ and a number k, such that c is well formed w.r.t. $|\vec{l}| - k$, and $\langle c, m, \vec{l}\rangle \to^n \langle \texttt{stop}, m', \vec{l'}\rangle$, there there exists c'', m'', $\vec{l''}$, α, k' and n', such that $\langle c, m, \vec{l}\rangle \curvearrowright_{\alpha}^{k'} \langle c'', m'', \vec{l''}\rangle$ and $\langle c'', m'', \vec{l''}\rangle \to^{n'} \langle \texttt{stop}, m', \vec{l'}\rangle$, where $k' + n' + 1 = n$.*

Proof. By induction on n.

3.7 Multiple step adequacy

In order to apply the above to Theorem 1, we need a way to convert between the indexed and unindexed multiple step relation. This is done with the following lemma.

Lemma 12 (Adequacy of multiple steps). *Given a command c, a memory m, and a stack $\vec{l}$, then $\langle c, m, \vec{l}\rangle \to^* \langle c', m', \vec{l'}\rangle$ if and only if there exists a n such that $\langle c, m, \vec{l}\rangle \to^n \langle c', m', \vec{l'}\rangle$.*

Proof. We look at the two directions seperatly. In both cases, it is proved på induction on the multiple step relation.

3.8 Revisiting Theorem 1

We are now ready to proof Theorem 1 by combining all of the pieces.

Restatement of Theorem 1 (Soundness of the dynamic semantics). *Given a program c such that $\langle c, m, \vec{l}\rangle \to^* \langle c', m', \vec{l'}\rangle$ then c satisfies Definition 2.*

Proof. Unfolding Definition 2, we need to show that for all pairs of memories m_1 and m_2, such that $m_1 \sim m_2$, and

$$\langle c, m_1, \vec{l}\rangle \to^* \langle \texttt{stop}, m_1', \vec{l'}\rangle$$

and

$$\langle c, m_2, \vec{l}\rangle \to^* \langle \texttt{stop}, m_2', \vec{l'}\rangle$$

it holds that $m_1' \sim m_2'$. By Lemma 12 there must be n_1 and n_2, such that

$$\langle c, m_1, \vec{l}\rangle \to^{n_1} \langle \texttt{stop}, m_1', \vec{l'}\rangle \quad \text{and} \quad \langle c, m_2, \vec{l}\rangle \to^{n_2} \langle \texttt{stop}, m_2', \vec{l'}\rangle$$

We can now proceed with strong induction on n_1. Here we apply Lemma 11 to both of the runs, using Lemma 2 on the two bridges, and applying the induction hypothesis to the two remaining multiple step relations.