

# Security Type Systems as Recursive Predicates<sup>\*</sup>

Andrei Popescu

Technische Universität München

**Abstract.** We show how security type systems from the literature of language-based noninterference can be represented more directly as predicates defined by structural recursion on the programs. In this context, we show how our uniform syntactic criteria from [7,8] cover several previous type-system soundness results.

## 1 Security type systems

As in Example 2 from [7, 8], we assume that atomic statements and tests are built by means of expressions applied to variables taken from a set **var**, ranged over by  $x, y, z$ . Thus, **exp**, ranged over by  $e$ , is the set of arithmetic expressions (e.g.,  $x + 1$ ,  $x * y + 5$ ). Then atomic commands  $atm \in \mathbf{atom}$  are assignment statements  $x := e$  and tests  $tst \in \mathbf{test}$  are Boolean expressions built from **exp** (e.g.,  $x > 0$ ,  $x + 1 = y + z$ ). For any expression  $e$  and test  $tst$ ,  $\text{Vars } e$  and  $\text{Vars } tst$  denote their sets of variables.

States are assignments of integers to variables, i.e., the set **state** is  $\mathbf{var} \rightarrow \mathbf{int}$ . Variables are classified as either low (lo) or high (hi) by a fixed security level function  $\text{sec} : \mathbf{var} \rightarrow \{\text{lo}, \text{hi}\}$ . We let  $L$  be the lattice  $\{\text{lo}, \text{hi}\}$ , where  $\text{lo} < \text{hi}$ .<sup>1</sup> We shall use the standard infima and suprema notations for  $L$ . Then  $\sim$  is defined as follows:  $s \sim t \equiv \forall x \in \mathbf{var}. \text{sec } x = \text{lo} \implies s \ x = t \ x$ .

We shall look into type systems from the literature, ::, assigning security levels  $l \in \{\text{lo}, \text{hi}\}$ , or pairs of security levels, to expressions and commands. All have in common the following:

Typing of expressions:

$$e :: \text{lo} \text{ if } \forall x \in \text{Vars } e. \text{sec } x = \text{lo} \qquad e :: \text{hi} \text{ always}$$

Typing of tests (similar):

$$tst :: \text{lo} \text{ if } \forall x \in \text{Vars } tst. \text{sec } x = \text{lo} \qquad tst :: \text{hi} \text{ always}$$

The various type systems shall differ in the typing of commands.

But first let us look more closely at their aforementioned common part. We note that, if an expression or a test has type  $l$  and  $l \leq k$ , then it also has type  $k$ . In other words, the following covariant subtyping rules for tests and expressions hold:

<sup>\*</sup> This work was supported by the DFG project Ni 491/13–2, part of the DFG priority program Reliably Secure Software Systems (RS3).

<sup>1</sup> One can also consider the more general case of *multilevel security*, via an unspecified lattice of security levels  $L$ —however, this brings neither much additional difficulty, nor much additional insight, so here focus on this 2-level lattice.

$$\frac{e :: l \quad l \leq k}{e :: k} (\text{SUBTYPE-EXP})$$

$$\frac{tst :: l \quad l \leq k}{tst :: k} (\text{SUBTYPE-TST})$$

Thus, the typing of an expression or test is uniquely determined by its *minimal type*, defined as follows:

$$\text{minTp } e = \bigvee \{ \text{sec } x. x \in \text{Vars } e \} \quad \text{minTp } tst = \bigvee \{ \text{sec } x. x \in \text{Vars } tst \}$$

The minimal typing operators can of course recover the original typing relation  $::$  as follows:

**Lemma 1.** The following hold:

- (1)  $e :: l$  iff  $\text{minTp } e \leq l$ .
- (2)  $tst :: l$  iff  $\text{minTp } tst \leq l$ .

### 1.1 Volpano-Smith possibilistic noninterference

In [11, §4], the typing of commands (which we denote by  $::_1$ ) is defined inductively as follows:

$$\frac{\text{sec } x = l \quad e :: l}{(x := e) ::_1 l} (\text{ASSIGN})$$

$$\frac{c_1 ::_1 l \quad c_2 ::_1 l}{(\text{Seq } c_1 c_2) ::_1 l} (\text{COMPOSE})$$

$$\frac{tst ::_1 l \quad c_1 ::_1 l \quad c_2 ::_1 l}{(\text{If } tst c_1 c_2) ::_1 l} (\text{IF})$$

$$\frac{tst ::_1 \text{lo} \quad c ::_1 l}{(\text{While } tst c) ::_1 \text{lo}} (\text{WHILE})$$

$$\frac{c_1 ::_1 l \quad c_2 ::_1 l}{(\text{Par } c_1 c_2) ::_1 l} (\text{PAR})$$

$$\frac{c ::_1 l \quad k \leq l}{c ::_1 k} (\text{SUBTYPE})$$

We think of  $c ::_1 l$  as saying:

- There is no downwards flow in  $c$ .
- $l$  is a lower bound on the level of the variables that the execution of  $c$  writes to.

(This intuition is accurately reflected by Lemma 2 below.)

Actually, [11] does not explicitly consider a rule like (PAR), and in fact uses parallel composition only at the top level. However, it does require that the thread pool (which can be viewed as consisting of a number of parallel compositions) has well-typed threads, which is the same as typing the pool to the minimum of the types of its threads—this is precisely what (PAR) does. (Also, in [11], the rule (WHILE) has the assumption  $c ::_1 \text{lo}$  rather than  $c ::_1 l$ —this alternative is of course equivalent, thanks to (SUBTYPE).)

Due to the subtyping rule, here we have a phenomenon dual to the one for expressions and tests: if a command has type  $l$  and  $k \leq l$ , then it also has type  $k$ —thus, the typing of a command, if any, is uniquely determined by its *maximal type*. The difference from expressions and tests is that such a type may not exist, making it necessary to keep a “safety” predicate during the computation of the maximal type. For example, consider the computation of the minimal type of  $\text{If } tst c_1 c_2$  according to the (IF) rule: Assume  $l_0$  is the minimal type of  $tst$  and  $l_1, l_2$  are the maximal types of  $c_1$  and  $c_2$ , respectively. The rule (IF) requires the three types involved in the hypothesis to be equal, and therefore

we need to *upcast*  $l_0$  and *downcast*  $l_1$  and  $l_2$  so that we obtain a common type  $l$ —thus, we need  $l_0 \leq l \leq l_1 \wedge l_2$ . Moreover,  $l$  has to be as high as possible. Such an  $l$  of course only exists if  $l_0 \leq l_1 \wedge l_2$ , and in this case the maximal  $l$  is  $l_1 \wedge l_2$ . In summary, the rule (IF) tells us the following:

- If  $tst\ c_1\ c_2$  is safe (i.e., type checks) iff  $c_1$  and  $c_2$  are safe and  $l_0 \leq l \leq l_1 \wedge l_2$ .
- If safe, the maximal type of  $tst\ c_1\ c_2$  is  $l_1 \wedge l_2$ .

Applying this reasoning to all the rules for  $::_1$ , we obtain the function  $\maxTp_1 : \mathbf{com} \rightarrow L$  and the predicate  $\mathsf{safe}_1 : \mathbf{com} \rightarrow \mathbf{bool}$  defined recursively on the structure of commands:<sup>2</sup>

**Definition 1.** –  $\mathsf{safe}_1\ (x := e) = (\minTp\ e \leq \sec\ x)$

- $\maxTp_1\ (x := e) = \sec\ x$
- $\mathsf{safe}_1\ (\text{Seq}\ c_1\ c_2) = (\mathsf{safe}_1\ c_1 \wedge \mathsf{safe}_1\ c_2)$
- $\maxTp_1\ (\text{Seq}\ c_1\ c_2) = (\maxTp_1\ c_1 \wedge \maxTp_1\ c_2)$
- $\mathsf{safe}_1\ (\text{If}\ tst\ c_1\ c_2) = (\mathsf{safe}_1\ c_1 \wedge \mathsf{safe}_1\ c_2 \wedge (\minTp\ tst \leq (\maxTp_1\ c_1 \wedge \maxTp_1\ c_2)))$
- $\maxTp_1\ (\text{If}\ tst\ c_1\ c_2) = (\maxTp_1\ c_1 \wedge \maxTp_1\ c_2)$
- $\mathsf{safe}_1\ (\text{While}\ tst\ c) = (\mathsf{safe}_1\ c \wedge (\minTp\ tst = \text{lo}))$
- $\maxTp_1\ (\text{While}\ tst\ c) = \text{lo}$
- $\mathsf{safe}_1\ (\text{Par}\ c_1\ c_2) = (\mathsf{safe}_1\ c_1 \wedge \mathsf{safe}_1\ c_2)$
- $\maxTp_1\ (\text{Par}\ c_1\ c_2) = (\maxTp_1\ c_1 \wedge \maxTp_1\ c_2)$

**Lemma 2.** The following are equivalent:

- (1)  $c ::_1 l$
- (2)  $\mathsf{safe}_1\ c$  and  $l \leq \maxTp_1\ c$ .

*Proof idea:* (1) implies (2): By easy induction on the definition of  $::_1$ .  
(2) implies (1): By easy structural induction on  $c$ . □

Now, let us write:

- $\text{low}\ e$ , for the sentence  $\minTp\ e = \text{lo}$
- $\text{low}\ tst$ , for the sentence  $\minTp\ tst = \text{lo}$
- $\text{fhigh}\ c$  (read “ $c$  finite and high”), for the sentence  $\maxTp_1\ c = \text{hi}$

(Thus,  $\text{low} : \mathbf{exp} \rightarrow \mathbf{bool}$ ,  $\text{low} : \mathbf{test} \rightarrow \mathbf{bool}$  and  $\text{fhigh} : \mathbf{com} \rightarrow \mathbf{bool}$ .)

Then, immediately from the definitions of  $\minTp$  and  $\maxTp_1$  (taking advantage of the fact that  $L = \{\text{hi}, \text{lo}\}$ ) we have the following:

- $\text{low}\ e = (\forall x \in \text{Vars}\ e. \sec\ x = \text{lo})$
- $\text{low}\ tst = (\forall x \in \text{Vars}\ tst. \sec\ x = \text{lo})$
- $\mathsf{safe}_1\ (x := e) = ((\sec\ x = \text{hi}) \vee \text{low}\ e)$
- $\text{fhigh}\ (x := e) = (\sec\ x = \text{hi})$
- $\mathsf{safe}_1\ (\text{Seq}\ c_1\ c_2) = (\mathsf{safe}_1\ c_1 \wedge \mathsf{safe}_1\ c_2)$

<sup>2</sup> Notice the overloaded, but consistent usage of the infimum operator  $\wedge$  in both the lattice  $L = \{\text{lo}, \text{hi}\}$  and the lattice of truth values  $\mathbf{bool}$  (the latter simply meaning the logical “and”).

- $\text{fhigh}(\text{Seq } c_1 \ c_2) = (\text{fhigh } c_1 \wedge \text{fhigh } c_2)$
- $\text{safe}_1(\text{If } \text{tst } c_1 \ c_2) = \begin{cases} \text{safe}_1 c_1 \wedge \text{safe}_1 c_2, & \text{if } \text{low } \text{tst} \\ \text{safe}_1 c_1 \wedge \text{safe } c_2 \wedge \text{fhigh } c_1 \wedge \text{fhigh } c_2, & \text{otherwise} \end{cases}$
- $\text{fhigh}(\text{If } \text{tst } c_1 \ c_2) = (\text{fhigh } c_1 \wedge \text{fhigh } c_2)$
- $\text{safe}_1(\text{While } \text{tst } c) = (\text{low } \text{tst} \wedge \text{safe}_1 c)$
- $\text{fhigh}(\text{While } \text{tst } c) = \text{False}$
- $\text{safe}_1(\text{Par } c_1 \ c_2) = (\text{safe}_1 c_1 \wedge \text{safe}_1 c_2)$
- $\text{low}(\text{Par } c_1 \ c_2) = (\text{low } c_1 \wedge \text{low } c_2)$

Notice that the above clauses characterize the prediactes  $\text{safe}_1 : \mathbf{com} \rightarrow \mathbf{bool}$  and  $\text{fhigh} : \mathbf{com} \rightarrow \mathbf{bool}$  uniquely, i.e., could act as their definitions (recursively on the structure of commands). Since the predicate  $\text{safe}_1$  is stronger than  $\text{fhigh}$  (as its clauses are strictly stronger), we can remove  $\text{safe}_1 c_1 \wedge \text{safe } c_2$  from the “otherwise” case of the *If* clause for  $\text{safe}_1$ , obtaining:

$$\text{safe}_1(\text{If } \text{tst } c_1 \ c_2) = \begin{cases} \text{safe}_1 c_1 \wedge \text{safe}_1 c_2, & \text{if } \text{low } \text{tst} \\ \text{fhigh } c_1 \wedge \text{fhigh } c_2, & \text{otherwise} \end{cases} = \begin{cases} \text{safe}_1 c_1 \wedge \text{safe}_1 c_2, & \text{if } \text{low } \text{tst} \\ \text{fhigh}(\text{If } \text{tst } c_1 \ c_2), & \text{otherwise} \end{cases}$$

The clauses for  $\text{safe}_1$  and  $\text{fhigh}$  are now seen to coincide with our [7, 8, §6] clauses for  $\overline{\approx_{\text{WT}}}$  and  $\text{discr} \wedge \text{may}\overline{\text{T}}$ , respectively, with the following variation: in [7, 8, §6] we do not commit to particular forms of tests or atomic statements, and therefore replace:

- $\text{low } \text{tst}$  with  $\text{cpt } \text{tst}$
- $\text{fhigh } \text{atm}$  with  $\text{pres } \text{atm}$  (where  $\text{atm}$  is an atom, such as  $x := e$ )
- $\text{safe}_1 \text{atm}$  with  $\text{cpt } \text{atm}$

Note that the predicates  $\text{cpt}$  and  $\text{pres}$ , as defined in [7, 8, §4], are semantic conditions expressed in terms of state indistinguishability, while  $\text{low}$ ,  $\text{fhigh}$  and  $\text{safe}_1$  are syntactic checks. than syntactic checks as here—the syntactic checks are easily seen to be stronger, i.e., we have  $\text{low } \text{tst} \implies \text{cpt } \text{tst}$ ,  $\text{fhigh } \text{atm} \implies \text{pres } \text{atm}$  and  $\text{safe}_1 \text{atm} \implies \text{cpt } \text{atm}$ .

The main concurrent noninterference result from [11], Corollary 5.7, states (something slightly weaker than) the following: if  $c ::_1 l$  for some  $l \in L$ , then  $c \approx_{\text{WT}} c$ . In the light of Lemma 2 and the above discussion, this result is subsumed by our Prop. 4 from [7, 8], taking  $\chi$  to be  $\approx_{\text{WT}}$ .

For the rest of the type systems we discuss, we shall proceed with similar transformations at a higher pace.

## 1.2 Volpano-Smith scheduler-independent noninterference

In [11, §7], another type system is defined,  $::_2$ , which has the same typing rules as  $::_1$  except for the rule for *If*, which is weakened by requiring the typing of the test to be  $\text{lo}$ .<sup>3</sup>

$$\frac{\text{tst} :: \text{lo} \quad c_1 ::_2 l \quad c_2 ::_2 l}{(\text{If } \text{tst } c_1 \ c_2) ::_2 l} (\text{IF})$$

<sup>3</sup> The same type system (except for the (PAR) rule) is introduced in [12] for a sequential language with the purpose of preventing leaks through the covert channels of termination and exceptions.

**Definition 2.** We define  $\text{safe}_2$  just like  $\text{safe}_1$ , except for the case of  $\text{If}$ , which becomes:

$$- \text{safe}_2 (\text{If } \text{tst } c_1 \ c_2) = ((\min \text{Tp } \text{tst} = \text{lo}) \wedge \text{safe}_2 c_1 \wedge \text{safe}_2 c_2)$$

Similarly to Lemma 2, we can prove:

**Lemma 3.** The following are equivalent:

- (1)  $c ::_2 l$
- (2)  $\text{safe}_2 c$  and  $l \leq \max \text{Tp}_1 c$ .

The inferred clauses for  $\text{safe}_2$  are the same as those for  $\text{safe}_1$ , except for the one for  $\text{If}$ , which becomes:

$$- \text{safe}_2 (\text{If } \text{tst } c_1 \ c_2) = (\text{low } \text{tst} \wedge \text{safe}_2 c_1 \wedge \text{safe}_2 c_2)$$

Then  $\text{safe}_2$  is seen to coincide with  $\overline{\text{siso}}$  from [7, 8, §6].

In [11] it is proved (via Theorem 7.1) that the soundness result for  $::_1$  also holds for  $::_2$ . In fact, one can see that Theorem 7.1 can be used to prove something much stronger: if  $c ::_2 l$  for some  $l \in L$ , then  $\text{siso } c$ . This result is subsumed by our Prop. 4 from [7, 8], taking  $\chi$  to be  $\text{siso}$ .

### 1.3 Boudol-Castellani termination-insensitive noninterference

As we already discussed in [7, 8], Boudol and Castellani [3, 4] work on improving the harsh Volpano-Smith typing of  $\text{While}$  (which requires low tests), but they pay a (comparatively small) price in terms of typing sequential composition, where what the first command reads is required to be below what the second command writes. (Essentially the same type system is introduced independently by Smith [9, 10] for studying probabilistic noninterference in the presence of uniform scheduling. Boudol and Castellani, as well as Smith, consider parallel composition only at the top level. Barthe and Nieto [1] raise this restriction, allowing nesting  $\text{Par}$  inside other language constructs, as we do here.)

To achieve this, they type commands  $c$  to a pair of security levels  $(l, l')$ : the contravariant “write” type  $l$  (similar to the Volpano-Smith one) and an extra covariant “read” type  $l'$ .

$$\begin{array}{c} \frac{\text{sec } x = l \quad e :: l}{(x := e) ::_2 (l, l')} (\text{ASSIGN}) \quad \frac{c_1 ::_3 (l_1, l'_1) \quad c_2 ::_3 (l_2, l'_2) \quad l'_1 \leq l_2}{(\text{Seq } c_1 \ c_2) ::_3 (l_1 \wedge l_2, l'_1 \vee l'_2)} (\text{COMPOSE}) \\[10pt] \frac{\text{tst} :: l_0 \quad c_1 ::_3 (l, l') \quad c_2 ::_3 (l, l') \quad l_0 \leq l}{(\text{If } \text{tst } c_1 \ c_2) ::_3 (l, l_0 \vee l')} (\text{IF}) \quad \frac{\text{tst} :: l' \quad c ::_3 (l, l') \quad l' \leq l}{(\text{While } \text{tst } c) ::_3 (l, l')} (\text{WHILE}) \\[10pt] \frac{c_1 ::_3 l \quad c_2 ::_3 l}{(\text{Par } c_1 \ c_2) ::_3 l} (\text{PAR}) \quad \frac{c ::_3 (l_1, l'_1) \quad l_2 \leq l_1 \quad l'_1 \leq l'_2}{c ::_3 (l_2, l'_2)} (\text{SUBTYPE}) \end{array}$$

We think of  $c ::_3 (l, l')$  as saying:

- There is no downwards flow in  $c$ .

- $l$  is a lower bound on the level of the variables that the execution of  $c$  writes to.
- $l'$  is an upper bound on the level of the variables that  $c$  reads, more precisely, that the control flow of the execution of  $c$  depends on.

(This intuition is accurately reflected by Lemma 4 below.)

In [3, 4], the rule for While is slightly different, namely:

$$\frac{tst :: l_0 \quad c ::_3 (l, l') \quad l_0 \vee l' \leq l}{(\text{While } tst \ c) ::_3 (l, l_0 \vee l')} \text{(WHILE')}$$

However, due to subtyping, it is easily seen to be equivalent to the one we listed. Indeed:

- (WHILE) is an instance of (WHILE') taking  $l_0 = l'$ .
- Conversely, (WHILE') follows from (WHILE) as follows: Assume the hypotheses of (WHILE'). By subtyping, we have  $tst :: l_0 \vee l'$  and  $c ::_3 (l, l_0 \vee l')$ , hence, by (WHILE), we have  $(\text{While } tst \ c) ::_3 (l, l_0 \vee l')$ , as desired.

Following for  $::_3$  the same technique as in the case of  $::_1$  and  $::_2$ , we define the functions  $\text{maxWtp} : \mathbf{com} \rightarrow L$  (read “maximum writing type”) and  $\text{minRtp} : \mathbf{com} \rightarrow L$  (read “minimum reading type”) and the predicate  $\text{safe}_3 : \mathbf{com} \rightarrow \mathbf{bool}$ :

**Definition 3.** –  $\text{safe}_3 (x := e) = (\text{minTp } e \leq \text{sec } x)$

- $\text{maxWtp } (x := e) = \text{sec } x$
- $\text{minRtp } (x := e) = \text{lo}$
- $\text{safe}_3 (\text{Seq } c_1 \ c_2) = (\text{safe}_3 \ c_1 \wedge \text{safe}_3 \ c_2 \wedge (\text{minRtp } c_1 \leq \text{maxWtp } c_2))$
- $\text{maxWtp } (\text{Seq } c_1 \ c_2) = (\text{maxWtp } c_1 \wedge \text{maxWtp } c_2)$
- $\text{minRtp } (\text{Seq } c_1 \ c_2) = (\text{minRtp } c_1 \vee \text{minRtp } c_2)$
- $\text{safe}_3 (\text{If } tst \ c_1 \ c_2) = (\text{safe}_3 \ c_1 \wedge \text{safe}_3 \ c_2 \wedge (\text{minTp } tst \leq (\text{maxWtp } c_1 \wedge \text{maxWtp } c_2)))$
- $\text{maxWtp } (\text{If } tst \ c_1 \ c_2) = (\text{maxWtp } c_1 \wedge \text{maxWtp } c_2)$
- $\text{minRtp } (\text{If } tst \ c_1 \ c_2) = (\text{minTp } tst \vee \text{minRtp } c_1 \vee \text{minRtp } c_2)$
- $\text{safe}_3 (\text{While } tst \ c) = (\text{safe}_3 \ c \wedge ((\text{minTp } tst \vee \text{minRtp } c) \leq \text{maxWtp } c))$
- $\text{maxWtp } (\text{While } tst \ c) = \text{maxWtp } c$
- $\text{minRtp } (\text{While } tst \ c) = (\text{minTp } tst \vee \text{minRtp } c)$
- $\text{safe}_3 (\text{Par } c_1 \ c_2) = (\text{safe}_3 \ c_1 \wedge \text{safe}_3 \ c_2)$
- $\text{maxWtp } (\text{Par } c_1 \ c_2) = (\text{maxWtp } c_1 \wedge \text{maxWtp } c_2)$
- $\text{minRtp } (\text{Par } c_1 \ c_2) = (\text{minRtp } c_1 \vee \text{minRtp } c_2)$

Furthermore, similarly to the cases of  $\text{safe}_1$  and  $\text{safe}_2$ , we have that:

**Lemma 4.** The following are equivalent:

- (1)  $c ::_3 (l, l')$
- (2)  $\text{safe}_3 \ c$  and  $l \leq \text{maxWtp } c$  and  $\text{minRtp } c \leq l'$ .

Now, let us write:

- $\text{high } c$ , for the sentence  $\text{maxWtp } c = \text{hi}$
- $\text{low } c$ , for the sentence  $\text{minRtp } c = \text{lo}$

Then, immediately from the definitions of  $\text{maxWtp}$  and  $\text{minRtp}$ , we have the following:

- $\text{safe}_3 (x := e) = ((\text{sec } x = \text{hi}) \vee \text{low } e)$
- $\text{high } (x := e) = (\text{sec } x = \text{hi})$
- $\text{low } (x := e) = \text{True}$
- $\text{safe}_3 (\text{Seq } c_1 c_2) = (\text{safe}_3 c_1 \wedge \text{safe}_3 c_2 \wedge (\text{low } c_1 \vee \text{high } c_2))$
- $\text{high } (\text{Seq } c_1 c_2) = (\text{high } c_1 \wedge \text{high } c_2)$
- $\text{low } (\text{Seq } c_1 c_2) = (\text{low } c_1 \wedge \text{low } c_2)$
- $\text{safe}_3 (\text{If } \text{tst } c_1 c_2) = (\text{safe}_3 c_1 \wedge \text{safe}_3 c_2 \wedge (\text{low } \text{tst} \vee (\text{high } c_1 \wedge \text{high } c_2)))$
- $\text{high } (\text{If } \text{tst } c_1 c_2) = (\text{high } c_1 \wedge \text{high } c_2)$
- $\text{low } (\text{If } \text{tst } c_1 c_2) = (\text{low } \text{tst} \wedge \text{low } c_1 \wedge \text{low } c_2)$
- $\text{safe}_3 (\text{While } \text{tst } c) = (\text{safe}_3 c \wedge ((\text{low } \text{tst} \wedge \text{low } c) \vee \text{high } c))$
- $\text{high } (\text{While } \text{tst } c) = \text{high } c$
- $\text{low } (\text{While } \text{tst } c) = (\text{low } \text{tst} \wedge \text{low } c)$
- $\text{safe}_3 (\text{Par } c_1 c_2) = (\text{safe}_3 c_1 \wedge \text{safe}_3 c_2)$
- $\text{high } (\text{Par } c_1 c_2) = (\text{high } c_1 \wedge \text{high } c_2)$
- $\text{low } (\text{Par } c_1 c_2) = (\text{low } c_1 \wedge \text{low } c_2)$

Then  $\text{high}$  and  $\text{low}$  are stronger than  $\text{safe}_3$ , and hence we can rewrite the  $\text{Seq}$ ,  $\text{If}$  and  $\text{While}$  clauses for  $\text{safe}_3$  as follows:

- $\text{safe}_3 (\text{Seq } c_1 c_2) = ((\text{low } c_1 \wedge \text{safe}_3 c_2) \vee (\text{safe}_3 c_1 \wedge \text{high } c_2))$
- $\text{safe}_3 (\text{If } \text{tst } c_1 c_2) = \begin{cases} \text{safe}_3 c_1 \wedge \text{safe}_3 c_2, & \text{if } \text{low } \text{tst} \\ \text{high } c_1 \wedge \text{high } c_2, & \text{otherwise} \end{cases} = \begin{cases} \text{safe}_3 c_1 \wedge \text{safe}_3 c_2, & \text{if } \text{low } \text{tst} \\ \text{high } (\text{If } \text{tst } c_1 c_2), & \text{otherwise} \end{cases}$
- $\text{safe}_3 (\text{While } \text{tst } c) = ((\text{low } \text{tst} \wedge \text{low } c) \vee \text{high } c) = (\text{low } (\text{While } \text{tst } c) \vee \text{high } (\text{While } \text{tst } c))$

The clauses for  $\text{safe}_3$ ,  $\text{high}$  and  $\text{low}$  are now seen to coincide with our [7, 8, §6] clauses for  $\overline{\approx}_{01}$  and  $\text{discr}$  and  $\text{siso}$ , respectively.

The main concurrent noninterference result from [3, 4] (Theorem 3.13 in [3] and Theorem 3.16 in [4]), states (something slightly weaker than) the following: if  $c ::_3 l$  for some  $l \in L$ , then  $c \approx_{01} c$ . In the light of Lemma 4 and the above discussion, this result is subsumed by our Prop. 4 from [7, 8], taking  $\chi$  to be  $\approx_{01}$ .

#### 1.4 Matos and Boudol's further improvement

Mantos and Boudol [2, 5, 6] study a richer language than the one we consider here, namely, an ML-like language. Moreover, they also consider a declassification construct. We shall ignore these extra features and focus on the restriction of their results to our simple while language. Moreover, they parameterize their development by a set of strongly terminating expressions (commands in our setting)—here we fix this set to be that of commands not containing while loops.

The type system  $::_4$  from [2, 5, 6] is based on a refinement of  $::_3$ , noticing that, as far as the reading type goes, one does not care about all variables a command reads (i.e., the variables that affect the control flow of its execution), but can restrict attention to those that may affect the *termination* of its execution.

The typing rules of  $::_4$  are identical to those of  $::_3$ , except for the  $\text{If}$  rule, which becomes:

$$\frac{tst :: l_0 \quad c_1 ::_3 (l, l') \quad c_2 ::_3 (l, l') \quad l_0 \leq l}{(\text{If } tst \ c_1 \ c_2) ::_3 (l, k)} \text{(IF)}$$

where  $k = \begin{cases} \text{lo}, & \text{if } c_1, c_2 \text{ do not contain While subexpressions} \\ l_0 \vee l', & \text{otherwise} \end{cases}$

We think of  $c ::_4 (l, l')$  as saying:

- There is no downwards flow in  $c$ .
- $l$  is a lower bound on the level of the variables that the execution of  $c$  writes to.
- $l'$  is an upper bound on the level of the variables that  $c$  termination-reads, i.e., that termination of the execution of  $c$  depends on.

(In [2, 5, 6], While is not a primitive, but is derived from higher-order recursion—however, the effect of the higher-order typing system on While is the same as that of our  $::_3$ , as shown in [6]. Moreover, due to working in a functional language with side effects, [2, 5, 6] record not two, but three security types: in addition to our  $l$  and  $l'$  (called there the writing and termination effects, respectively), they also record  $l''$  (called there the reading effect) which represents an upper bound on the security levels of variables the returned value of  $c$  depends on—here, this information is unnecessary, since  $c$  returns no value.)

**Definition 4.** We define the function  $\text{minTRtp} : \mathbf{com} \rightarrow L$  (read “minimum termination-reading type”) and the predicate  $\text{safe}_4 : \mathbf{com} \rightarrow \mathbf{bool}$  as follows:  $\text{minTRtp}$  is defined using the same recursive clauses as  $\text{minRtp}$ , except for the clause for If, which becomes:

- $\text{minTRtp} (\text{If } tst \ c_1 \ c_2) = \begin{cases} \text{lo}, & \text{if } c_1, c_2 \text{ do not contain While subexpressions} \\ \text{minTp } tst \vee \text{minTRtp } c_1 \vee \text{minTRtp } c_2, & \text{otherwise} \end{cases}$

$\text{safe}_4$  is defined using the same clauses as  $\text{safe}_3$  with  $\text{minTRtp}$  replacing  $\text{minRtp}$ .

**Lemma 5.** The following are equivalent:

- (1)  $c ::_4 (l, l')$
- (2)  $\text{safe}_4 \ c$  and  $l \leq \text{maxWtp } c$  and  $\text{minTRtp } c \leq l'$ .

Now, let us write:

- $\text{wlow } c$  (read “ $c$  has low tests on top of while subexpressions”), for the sentence  $\text{minTRtp } c = \text{lo}$
- $\text{noWhile } c$ , for the sentence “ $c$  contains no While subexpressions”

We obtain:

- $\text{safe}_4 (x := e) = ((\text{sec } x = \text{hi}) \vee \text{low } e)$
- $\text{wlow } (x := e) = \text{True}$
- $\text{safe}_4 (\text{Seq } c_1 \ c_2) = (\text{safe}_4 \ c_1 \wedge \text{safe}_4 \ c_2 \wedge (\text{wlow } c_1 \vee \text{high } c_2))$
- $\text{wlow } (\text{Seq } c_1 \ c_2) = (\text{wlow } c_1 \wedge \text{wlow } c_2)$
- $\text{safe}_4 (\text{If } tst \ c_1 \ c_2) = (\text{safe}_4 \ c_1 \wedge \text{safe}_4 \ c_2 \wedge (\text{wlow } tst \vee (\text{high } c_1 \wedge \text{high } c_2)))$



- $\text{wlow} (\text{If } \text{tst } c_1 \ c_2) = (\text{low } \text{tst} \wedge \text{wlow } c_1 \wedge \text{wlow } c_2) \vee (\text{noWhile } c_1 \wedge \text{noWhile } c_2)$
- $\text{safe}_4 (\text{While } \text{tst } c) = (\text{safe}_4 c \wedge ((\text{low } \text{tst} \wedge \text{low } c) \vee \text{high } c))$
- $\text{wlow} (\text{While } \text{tst } c) = (\text{low } \text{tst} \wedge \text{wlow } c)$
- $\text{safe}_4 (\text{Par } c_1 \ c_2) = (\text{safe}_4 c_1 \wedge \text{safe}_4 c_2)$
- $\text{wlow} (\text{Par } c_1 \ c_2) = (\text{wlow } c_1 \wedge \text{wlow } c_2)$

We can prove by induction on  $c$  that  $\text{safe}_1 c = (\text{safe}_4 c \wedge \text{wlow } c)$ . Using this, we rewrite the Seq, If and While clauses for  $\text{safe}_4$  as follows:

- $\text{safe}_4 (\text{Seq } c_1 \ c_2) = ((\text{safe}_1 c_1 \wedge \text{safe}_4 c_2) \vee (\text{safe}_4 c_1 \wedge \text{high } c_2))$
- $\text{safe}_4 (\text{If } \text{tst } c_1 \ c_2) = \begin{cases} \text{safe}_4 c_1 \wedge \text{safe}_4 c_2, & \text{if low } \text{tst} \\ \text{high } (\text{If } \text{tst } c_1 \ c_2), & \text{otherwise} \end{cases}$
- $\text{safe}_4 (\text{While } \text{tst } c) = (\text{safe}_1 (\text{While } \text{tst } c) \vee \text{high } (\text{While } \text{tst } c))$

Then  $\text{safe}_4$  turns out to coincide with our  $\overline{\approx_W}$  from [7, 8, §6].

The main noninterference result from [2, 5, 6] (in [2], the soundness theorem in §5), states the following: if  $c ::_4 l$  for some  $l \in L$ , then  $c \approx_W c$ . In the light of Lemma 4 and the above discussion, this result is subsumed by our Prop. 4 from [7, 8], taking  $\chi$  to be  $\approx_W$ .

## References

1. G. Barthe and L. P. Nieto. Formally verifying information flow type systems for concurrent and thread systems. In *FMSE*, pages 13–22, 2004.
2. G. Boudol. On typing information flow. In *ICTAC*, pages 366–380, 2005.
3. G. Boudol and I. Castellani. Noninterference for concurrent programs. In *ICALP*, pages 382–395, 2001.
4. G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002.
5. A. A. Matos and G. Boudol. On declassification and the non-disclosure policy. In *CSFW*, pages 226–240, 2005.
6. A. A. Matos and G. Boudol. On declassification and the non-disclosure policy. *Journal of Computer Security*, 17(5):549–597, 2009.
7. A. Popescu, J. Hölzl, and T. Nipkow. Proving concurrent noninterference. In *CPP*, pages 109–125, 2012.
8. A. Popescu, J. Hölzl, and T. Nipkow. Formal verification of concurrent noninterference. *Journal of Formalized Reasoning*, 2013. Extended version of [7]. To appear.
9. G. Smith. A new type system for secure information flow. In *IEEE Computer Security Foundations Workshop*, pages 115–125, 2001.
10. G. Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *IEEE Computer Security Foundations Workshop*, pages 3–13, 2003.
11. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *ACM Symposium on Principles of Programming Languages*, pages 355–364, 1998.
12. D. M. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *CSFW*, pages 156–169, 1997.