



Semantics and Verification of Software

Winter Semester 2017/18

Lecture 19: Security Type Systems

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<http://moves.rwth-aachen.de/teaching/ws-1718/sv-sw/>

Recap: Type Systems

Typed Execution of Statements

Definition (Typed execution of statements (cf. Definition 16.4))

$\rightarrow_1 \subseteq (Cmd \times \Sigma) \times (Cmd \times \Sigma)$ is given by:

$$\begin{array}{c} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \qquad \frac{\langle a, \sigma \rangle \rightarrow v}{\langle x := a, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto v] \rangle} \\ \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle} \qquad \frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow_1 \langle c'_1; c_2, \sigma' \rangle} \\ \dots \qquad \dots \end{array}$$

Thus: execution gets stuck iff expression evaluation gets stuck

Example

For $c = (x := x + 1)$,

- if $\sigma(x) = 2 \in \mathbb{Z}$, then $\langle c, \sigma \rangle \rightarrow_1 \sigma[x \mapsto 3]$
- if $\sigma(x) = 3.14 \in \mathbb{R}$, then $\langle c, \sigma \rangle \not\rightarrow_1$

Recap: Type Systems

Data Types and Type Environments

- Data types

$$Typ = \{int, real\}$$

with $\tau \in Typ$

- Observation: type of (arithmetic) expression depends on types of contained variables
- Therefore: introduce **type environments**

$$TEnv = \{\Gamma \mid \Gamma : Var \rightarrow Typ\}$$

- In the following: typing rule systems for

- arithmetic expression $a \in AExp$ has type $\tau \in Typ$:
- Boolean expression $b \in BExp$ is well-typed:
- statement $c \in Cmd$ is well-typed:

$$\Gamma \vdash a : \tau$$

$$\Gamma \vdash b$$

$$\Gamma \vdash c$$

in type environment $\Gamma \in TEnv$

Recap: Type Systems

Typing Rules for Commands

Definition (Well-typedness of commands)

For $\Gamma \in TEnv$ and $c \in Cmd$, $\Gamma \vdash c$ is given by

$$\frac{}{\Gamma \vdash \text{skip}} \quad \frac{\Gamma \vdash a : \Gamma(x)}{\Gamma \vdash x := a} \quad \frac{\Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash c_1 ; c_2}$$
$$\frac{\Gamma \vdash b \quad \Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}} \quad \frac{\Gamma \vdash b \quad \Gamma \vdash c}{\Gamma \vdash \text{while } b \text{ do } c \text{ end}}$$

Thus: command not well-typed iff some expression not well-typed or assignment incompatible

Recap: Type Systems

Type Soundness for Commands

Theorem (Progress)

If $c \in \text{Cmd} \setminus \{\downarrow\}$, $\sigma \in \Sigma$ and $\Gamma \in \text{TEnv}$ such that $\Gamma \vdash c$ and $\Gamma \vdash \sigma$, then there exist $c' \in \text{Cmd}$ and $\sigma' \in \Sigma$ such that $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$.

Theorem (Preservation)

If $c, c' \in \text{Cmd}$, $\sigma, \sigma' \in \Sigma$ and $\Gamma \in \text{TEnv}$ such that $\Gamma \vdash c$, $\Gamma \vdash \sigma$ and $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$, then $\Gamma \vdash c'$ and $\Gamma \vdash \sigma'$.

Corollary (Type soundness)

If $c, c' \in \text{Cmd}$, $\sigma, \sigma' \in \Sigma$ and $\Gamma \in \text{TEnv}$ such that $\Gamma \vdash c$, $\Gamma \vdash \sigma$ and $\langle c, \sigma \rangle \rightarrow_1^ \langle c', \sigma' \rangle$ with $c' \neq \downarrow$, then there exist $c'' \in \text{Cmd}$ and $\sigma'' \in \Sigma$ such that $\langle c', \sigma' \rangle \rightarrow_1 \langle c'', \sigma'' \rangle$.*

All proofs by rule induction (based on the corresponding results for expressions)

Language-Based Security

Information flow control

Goal: Ensure that programs protect private data like passwords, bank details, or medical records. There should be no information flow from private data into public channels.

Language-based security

Approach to information flow control where **data flow analysis** is used to determine whether a program is free of illicit information flows

- guarantees confidentiality by **program analysis**, not by cryptography
- analyses often expressed as **type systems**

Security Levels

- Data objects (such as program variables) have **security/confidentiality levels**
- Security levels are **partially ordered**:

$l < l'$ means that l is less confidential than l'

(thus, information is allowed to flow from l to l' but not vice versa)

- Simplest choice: **low** (low/public) and **high** (high/confidential/secret/private) with $\text{low} < \text{high}$
- Extension to multi-level security by generalisation to **lattice** SL
- Security levels of variables given by $sl : Var \rightarrow SL$

Example 19.1 (Illicit flows)

Let $sl(\text{low}) = \text{low}$ and $sl(\text{high}) = \text{high}$.

Explicit: `low := high`

Implicit (“occlusion”): `if high = 1 then low := 1 else low := 2 end`

Security Levels on Expressions

The security level of an expression is the maximal security level of any of its variables.

Definition 19.2 (Security levels on expressions)

$sl : Var \rightarrow SL$ is extended to $sl : AExp \cup BExp \rightarrow SL$ as follows:

$$\begin{array}{ll} sl(z) = \min SL & sl(t) = \min SL \\ sl(a_1 + a_2) = sl(a_1) \sqcup sl(a_2) & sl(a_1 = a_2) = sl(a_1) \sqcup sl(a_2) \\ \dots & sl(\neg b) = sl(b) \\ & sl(b_1 \wedge b_2) = sl(b_1) \sqcup sl(b_2) \\ & \dots \end{array}$$

Security Levels on States

Idea: identify when states can (not) be distinguished by a “low observer”

Definition 19.3 (Security levels on states)

Two program states $\sigma_1, \sigma_2 \in \Sigma$ **agree up to level** $l \in SL$ if for all $x \in Var$ such that $sl(x) \leq l$,

$$\sigma_1(x) = \sigma_2(x).$$

Notation: $\sigma_1 =_{\leq l} \sigma_2$ (analogously: $\sigma_1 =_{< l} \sigma_2$)

Non-Interference Informally

To guarantee confidentiality, high variables must not interfere with low ones, i.e., a variation of confidential input must not cause a variation of public output.

Non-interference (informally)

Program c guarantees **non-interference** if for all states σ_1, σ_2 : If σ_1 and σ_2 agree on low variables (but possibly differ on high variables), then the states resulting from executing $\langle c, \sigma_1 \rangle$ and $\langle c, \sigma_2 \rangle$ must also agree on low variables.

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \text{Cmd}$ guarantees **non-interference** at security level $l \in \text{SL}$ if for all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2 \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \rightarrow \sigma'_1, \langle c, \sigma_2 \rangle \rightarrow \sigma'_2 \Rightarrow \sigma'_1 =_{\leq l} \sigma'_2$$

Thus: every program c guarantees non-interference at level $l_c = \bigsqcup \{\text{sl}(x) \mid x \in \text{Var}_c\}$

Example 19.5 (Interference (cf. Example 19.1))

1. $c_1 = (\text{low} := \text{high})$ does not guarantee non-interference at level **low**:
 - let $\sigma_i = [\text{low} \mapsto 0, \text{high} \mapsto i]$ for $i = 1, 2$ (and thus $\sigma_1 =_{\leq \text{low}} \sigma_2$)
 - then $\langle c_1, \sigma_1 \rangle \rightarrow \sigma'_1$ and $\langle c_1, \sigma_2 \rangle \rightarrow \sigma'_2$ with $\sigma'_i(\text{low}) = i$ ($i = 1, 2$)
 - thus $\sigma'_1 \neq_{\leq \text{low}} \sigma'_2$.
2. Similarly: $c_2 = (\text{if high} = 1 \text{ then low} := 1 \text{ else low} := 2 \text{ end})$ does not guarantee non-interference at level **low**
3. Both programs guarantee non-interference at level **high**

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees **non-interference for well-typed programs**
- **Explicit flows** easy to handle: for each assignment $x := a$, check that $sl(x) \geq sl(a)$
- Approach to deal with **implicit flows**: parametrise typing rules with security level of Boolean expressions (“if **high** = 1”) that guard the current command (“then **low** := 1”)
- **Well-typedness predicate** for $l \in SL$ and $c \in Cmd$:

$$l \vdash c$$

Meaning: “In the context of Boolean expressions of level (at most) l , command c is well-typed.”

- Hence: assignments to variables of level $< l$ are **forbidden** in c

A Security Type System

The Security Type System

Definition 19.6 (Well-typedness of commands)

For $l \in SL$ and $c \in Cmd$, $l \vdash c$ is given by

$$\begin{array}{c} \frac{}{(s\text{-skip}) l \vdash \text{skip}} \quad \frac{sl(x) \geq sl(a) \quad sl(x) \geq l}{(s\text{-asgn}) l \vdash x := a} \quad \frac{l \vdash c_1 \quad l \vdash c_2}{(s\text{-seq}) l \vdash c_1 ; c_2} \\ \frac{l \sqcup sl(b) \vdash c_1 \quad l \sqcup sl(b) \vdash c_2}{(s\text{-if}) l \vdash \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}} \quad \frac{l \sqcup sl(b) \vdash c}{(s\text{-wh}) l \vdash \text{while } b \text{ do } c \text{ end}} \end{array}$$

Example 19.7

Let $SL = \{\text{low}, \text{high}, \text{secret}\}$ (on the board) with $\text{low} < \text{high} < \text{secret}$.

1. $\text{low} \vdash \text{if high} = \text{low} \text{ then high} := 1 \text{ else skip end}$
2. $\text{low} \not\vdash \text{if high} = \text{low} \text{ then low} := 1 \text{ else skip end}$
3. $\text{high} \vdash \text{if high} = \text{low} \text{ then high} := 1 \text{ else skip end}$
4. $\text{secret} \not\vdash \text{if high} = \text{low} \text{ then high} := 1 \text{ else skip end}$

A Security Type System

Confinement

The following lemma states that if $l \vdash c$, then c cannot modify variables of level $< l$, i.e., the effect of c is **confined** to variables of level $\geq l$:

Lemma 19.8 (Confinement)

Let $c \in \text{Cmd}$ and $l \in \text{SL}$ such that $l \vdash c$. Then for all $\sigma, \sigma' \in \Sigma$ with $\langle c, \sigma \rangle \rightarrow \sigma'$,

$$\sigma =_{<l} \sigma'$$

Proof.

omitted □

A Security Type System

Soundness

The following theorem states that the security type system is sound, i.e., that well-typedness guarantees non-interference at all levels:

Theorem 19.9 (Non-interference)

Let $c \in \text{Cmd}$ such that $\text{min } SL \vdash c$, and let $l \in SL$ and $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2 \in \Sigma$ such that $\sigma_1 =_{\leq l} \sigma_2$, $\langle c, \sigma_1 \rangle \rightarrow \sigma'_1$, and $\langle c, \sigma_2 \rangle \rightarrow \sigma'_2$. Then $\sigma'_1 =_{\leq l} \sigma'_2$.

Proof.

omitted □

The following example shows that the type system is incomplete, i.e., rejects some programs that guarantee non-interference:

Example 19.10

```
low  $\not\vdash$  if high = 1 then low := 1 else low := 1 end
```

A Security Type System

Extensions

- Does non-interference really guarantee absence of information flow?
 - problem: **covert channels**
 - example: $c = \text{while high} = 1 \text{ do skip end}$
 - then $\text{min } SL \vdash c$
 - but non-termination of c allows to conclude that $\text{high} = 1$
- Other kinds of covert channels: processing time, power consumption, ...
- **Encryption** breaks traditional non-interference as (public) ciphertexts do depend on (confidential) contents
 - example: $\text{low} := \text{encrypt}(\text{high})$
 - then $\sigma_1(\text{high}) \neq \sigma_2(\text{high})$ (usually) implies $\sigma'_1(\text{low}) \neq \sigma'_2(\text{low})$ after assignment
 - thus $\sigma'_1 \not\leq_{\text{low}} \sigma'_2$ (interference)
- Solution: relaxed notions of (non-)interference that distinguish between
 - breaking non-interference because of **legitimate use of (sufficiently strong) encryption** and
 - breaking non-interference due to an **(unintended) leak**(“possibilistic non-interference”)
- Excellent survey paper:
 - A. Sabelfeld, A.C. Myers: Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications, 21(1), 2003, 5–19*