Logics in Security **Winter 2014**

Homework for Module 4

Instructor: Deepak Garg TA: Iulia Boloşteanu
dg@mpi-sws.org iulia_mb@mpi-sws.org

Release date: 02.02.2015 *Due date: 14.02.2015*

General instructions: Attempt all questions. Submit your homework via email to both the instructor and the TA before midnight on the due date. The LaTeX source for this homework will be provided to help you typeset. You can also typeset using any other means, including simple ASCII.

Homework instructions: This homework is divided into two sections. Section 1 tests your understanding of programming language semantics, typing and type-safety, as taught in class. Section 2 tests your understanding of the type system for information flow from the paper of Volpano, Smith and Irvine.

1 Operational Semantics and Types

For this section, refer to the operational semantics in Appendix A and the typing rules in Appendix B. Appendix C is **not** to be referred for this section.

Problem 4-1 (5 points)

Expression determinism lemma. A key lemma in the proof of type-safety that we discussed in class is determinism of expression evaluation. Prove that lemma here: If $\mu, e \Downarrow v_1$ and $\mu, e \Downarrow v_2$ then $v_1 = v_2$.
State clearly what you are are inducting on. Independent of what you induct on, you only need to write the induction cases corresponding to the following forms of e: v, l, and $e_1 + e_2$.

Problem 4-2 (5 points)

Progress theorem. The progress theorem says that a well-typed program that is not skip must always reduce further. Formally, if $\lambda \vdash p$ and $\lambda \vdash \mu$ and $p \neq \mathsf{skip}$, then there exist μ', p' such that $\mu, p \to \mu', p'$.
The proof proceeds by induction on the derivation of $\lambda \vdash p$. In class, we covered all cases of the proof except $p = \mathsf{while}\ e\ \mathsf{do}\ p'$. Show that case of the proof.

Problem 4-3 (5 points)

Type-safety. Consider the program $p = (\ell := \mathsf{true}; \mathsf{if}\ \ell\ \mathsf{then}\ \ell := \mathsf{false}\ \mathsf{else}\ \ell := 1)$. Assuming the typing context is $\lambda = \{\ell : \mathtt{int}\}$, answer the following questions:

1. Is p well-typed, i.e., is $\lambda \vdash p$ provable? Justify your answer.

2. Is p unsafe, i.e., can p reach a stuck (bad) state by reduction starting from any memory μ? (Recall that a program is bad/stuck when it is not skip and no reduction is possible.)

2 Types for information flow control

For this section, refer to the operational semantics in Appendix A and the typing rules in Appendix C. Appendix B is **not** to be referred for this section.

Problem 4-4 (10 points)

Lattices. The following exercise is designed to help you better understand the structure of lattices.

Definition 1. *(Complete lattice) A partially ordered set $(S, \sqsubseteq)$ is called a* complete lattice *if every subset M of S has a least upper bound and a greatest lower bound in $(S, \sqsubseteq)$.*

The least upper bound (also called lub or join) of two elements $a, b \in L$ is written $a \sqcup b$. The greatest lower bound (glb or meet) of two elements $a, b \in L$ is written $a \sqcap b$.

1. On natural numbers other than 0, consider the following order: $a \sqsubseteq b$ if and only if a divides b, i.e., $b \bmod a = 0$. For each of the following sets S, is $(S, \sqsubseteq)$ a complete lattice? Justify your answers.
   (a) $S = \mathbb{N}\backslash 0$ (all positive natural numbers).
   (b) $S = \{1, 2, 3, 4, 12, 24, 36, 48\}$.
   (c) $S = \{k \mid k \text{ divides } n\}$ where n is a fixed integer.
   (d) $S = \{k \mid n \text{ divides } k\}$ where n is a fixed integer.
2. Consider the lattice defined in 1(b) above. Calculate the following joins and meets: $2 \sqcup 3$, $12 \sqcup 24$, $2 \sqcap 3$, $24 \sqcup 36$, $48 \sqcap 36$, $12 \sqcap 3$.

Problem 4-5 (7 points)

Information flow. Let λ be a typing context — an assignment of lattice elements to memory locations. Recall that a program p is non-interfering (secure in the sense that it does not have any bad information flows) if the following hold for all $\tau, \mu_1, \mu_2, \mu_1', \mu_2'$: If for all l such that $\lambda(l) \sqsubseteq \tau$, $\mu_1(l) = \mu_2(l)$ and, additionally, $\mu_1, p \Rightarrow \mu_1'$ and $\mu_2, p \Rightarrow \mu_2'$, then for all l such that $\lambda(l) \sqsubseteq \tau$, $\mu_1'(l) = \mu_2'(l)$.

1. Let the lattice be $L \sqsubseteq H$ and let $\lambda = \{x : L, y : H, z : H\}$. Consider the following program:

```
if (y = 1) then {
   x := 1;
}
else {
   z := y + 1;
}
x := 1;
```

   a. Assuming the initial memory is $\mu = \{x \mapsto 5, y \mapsto 1, z \mapsto 2\}$, what is the final memory μ' after the execution of the program?

b. Using the type system of Appendix C, derive a type of the form τ cmd for the program. If no type can be derived, explain where the derivation fails and why the program cannot be typed.

c. Is this program non-interfering? Justify your answer.

2. Let the lattice be $\mathcal{L} = \{L \sqsubseteq M_1, L \sqsubseteq M_2, M_1 \sqsubseteq H, M_2 \sqsubseteq H\}$ and let $\lambda = \{a : L, b : M_1, c : M_2, d : H\}$. Consider the following program:

$$
\begin{array}{l}
\textsf{while } (a > 0) \textsf{ do } \{ \\
\quad b := b + a; \\
\quad a := a - 1; \\
\} \\
\textsf{if } (b + 2 == 0) \textsf{ then} \\
\quad d := d + b; \\
\textsf{else} \\
\quad d := d + c;
\end{array}
$$

a. Using the type system of Appendix C, derive a type of the form τ cmd for the program. If no type can be derived, explain where the derivation fails and why the program cannot be typed.

b. Is this program non-interfering? Justify your answer.

A Syntax and Operational Semantics

Syntax:

Values	v	$::=$	$0 \mid 1 \mid 2 \ldots \mid \textsf{true} \mid \textsf{false}$
Expressions	e	$::=$	$v \mid \ell \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 == e_2 \mid e_1 > e_2$
Commands	p	$::=$	$\textsf{skip} \mid \ell := e \mid p_1; p_2 \mid \textsf{if } (e) \textsf{ then } p_1 \textsf{ else } p_2 \mid \textsf{while } (e) \textsf{ do } p$
Memory	μ	$::=$	$\ell_1 \mapsto v_1, \ldots, \ell_n \mapsto v_n$

Semantic rules for expressions:

Note: For any connective o in $\{+, -, ==, >\}$, $\hat{o}$ denotes the underlying arithmetic operator.

$$\frac{}{\mu, v \Downarrow v} \qquad \frac{}{\mu, \ell \Downarrow \mu(\ell)} \qquad \frac{\mu, e_1 \Downarrow v_1 \qquad \mu, e_2 \Downarrow v_2 \qquad v_1 \hat{+} v_2 = v}{\mu, e_1 + e_2 \Downarrow \mu, v}$$

$$\frac{\mu, e_1 \Downarrow v_1 \qquad \mu, e_2 \Downarrow v_2 \qquad v_1 \hat{-} v_2 = v}{\mu, e_1 - e_2 \Downarrow \mu, v} \qquad \frac{\mu, e_1 \Downarrow v_1 \qquad \mu, e_2 \Downarrow v_2 \qquad (v_1 \hat{==} v_2) = v}{\mu, e_1 == e_2 \Downarrow \mu, v}$$

$$\frac{\mu, e_1 \Downarrow v_1 \qquad \mu, e_2 \Downarrow v_2 \qquad (v_1 \hat{>} v_2) = v}{\mu, e_1 > e_2 \Downarrow \mu, v}$$

Small-step or reduction semantics for commands:

$$\frac{\mu, e \Downarrow v}{\mu, \ell := e \to \mu[\ell \mapsto v], \mathsf{skip}} \qquad \frac{\mu, p_1 \to \mu', p_1'}{\mu, p_1; p_2 \to \mu', p_1'; p_2} \qquad \frac{}{\mu, (\mathsf{skip}; p) \to \mu, p}$$

$$\frac{\mu, e \Downarrow \mathsf{true}}{\mu, \mathsf{if}\ (e)\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 \to \mu, p_1} \qquad \frac{\mu, e \Downarrow \mathsf{false}}{\mu, \mathsf{if}\ (e)\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 \to \mu, p_2}$$

$$\frac{\mu, e \Downarrow \mathsf{true}}{\mu, \mathsf{while}\ (e)\ \mathsf{do}\ p \to \mu, p; \mathsf{while}\ (e)\ \mathsf{do}\ p} \qquad \frac{\mu, e \Downarrow \mathsf{false}}{\mu, \mathsf{while}\ (e)\ \mathsf{do}\ p \to \mu, \mathsf{skip}}$$

Big-step semantics for commands:

$$\frac{}{\mu, \mathsf{skip} \Rightarrow \mathsf{skip}} \qquad \frac{\mu, e \Downarrow v}{\mu, \ell := e \Rightarrow \mu[\ell \mapsto v]} \qquad \frac{\mu, p_1 \Rightarrow \mu' \qquad \mu', p_2 \Rightarrow \mu''}{\mu, (p_1; p_2) \Rightarrow \mu''}$$

$$\frac{\mu, e \Downarrow \mathsf{true} \qquad \mu, p_1 \Rightarrow \mu'}{\mu, \mathsf{if}\ e\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 \Rightarrow \mu'} \qquad \frac{\mu, e \Downarrow \mathsf{false} \qquad \mu, p_2 \Rightarrow \mu'}{\mu, \mathsf{if}\ e\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 \Rightarrow \mu'}$$

$$\frac{\mu, e \Downarrow \mathsf{true} \qquad \mu, (p; \mathsf{while}\ e\ \mathsf{do}\ p) \Rightarrow \mu'}{\mu, \mathsf{while}\ e\ \mathsf{do}\ p \Rightarrow \mu'} \qquad \frac{\mu, e \Downarrow \mathsf{false}}{\mu, \mathsf{while}\ e\ \mathsf{do}\ p \Rightarrow \mu}$$

B Typing rules

$$\begin{array}{lrcl} \text{Types} & \tau & ::= & \texttt{int} \mid \texttt{bool} \\ \text{Type context} & \lambda & ::= & \ell_1 : \tau_1, \ldots, \ell_n : \tau_n \end{array}$$

Typing rules for expressions:

$$\frac{n \in \{0, 1, 2, \ldots\}}{\lambda \vdash n : \texttt{int}} \qquad \frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\lambda \vdash b : \texttt{bool}} \qquad \frac{\ell : \tau \in \lambda}{\lambda \vdash \ell : \tau} \qquad \frac{\lambda \vdash e_1 : \texttt{int} \qquad \lambda \vdash e_2 : \texttt{int}}{\lambda \vdash e_1 + e_2 : \texttt{int}}$$

$$\frac{\lambda \vdash e_1 : \texttt{int} \qquad \lambda \vdash e_2 : \texttt{int}}{\lambda \vdash e_1 - e_2 : \texttt{int}} \qquad \frac{\lambda \vdash e_1 : \texttt{int} \qquad \lambda \vdash e_2 : \texttt{int}}{\lambda \vdash e_1 == e_2 : \texttt{bool}}$$

$$\frac{\lambda \vdash e_1 : \texttt{int} \qquad \lambda \vdash e_2 : \texttt{int}}{\lambda \vdash e_1 > e_2 : \texttt{bool}}$$

Typing rules for commands:

$$\frac{}{\lambda \vdash \mathsf{skip}} \qquad \frac{\ell : \tau \in \lambda \qquad \lambda \vdash e : \tau}{\lambda \vdash \ell := e} \qquad \frac{\lambda \vdash p_1 \qquad \lambda \vdash p_2}{\lambda \vdash p_1; p_2}$$

$$\frac{\lambda \vdash e : \texttt{bool} \qquad \lambda \vdash p_1 \qquad \lambda \vdash p_2}{\lambda \vdash \mathsf{if}\ e\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2} \qquad \frac{\lambda \vdash e : \texttt{bool} \qquad \lambda \vdash p}{\lambda \vdash \mathsf{while}\ e\ \text{do}\ p}$$

Typing rule for memories:

$$\frac{\lambda \vdash v_1 : \lambda(\ell_1) \qquad \ldots \qquad \lambda \vdash v_n : \lambda(\ell_n)}{\lambda \vdash \ell_1 \mapsto v_1, \ldots, \ell_n \mapsto v_n}$$

C Types for Information Flow Control

For information flow control, types τ are elements of a lattice $(S, \sqsubseteq)$. A type context λ assigns types to memory locations. This is written $\lambda ::= \ell_1 : \tau_1, \ldots, \ell_n : \tau_n$.

Typing rules for expressions:

$$\frac{}{\lambda \vdash v : \tau} \qquad \frac{\ell : \tau \in \lambda}{\lambda \vdash \ell : \tau} \qquad \frac{\lambda \vdash e_1 : \tau_1 \qquad \lambda \vdash e_2 : \tau_2 \qquad \circ \in \{+, -, ==, >\}}{\lambda \vdash e_1 \circ e_2 : \tau_1 \sqcup \tau_2}$$

$$\frac{\lambda \vdash e : \tau \qquad \tau \sqsubseteq \tau'}{\lambda \vdash e : \tau'}$$

Typing rules for commands:

$$\frac{}{\lambda \vdash \mathsf{skip} : \tau\ \mathsf{cmd}} \qquad \frac{\ell : \tau \in \lambda \qquad \lambda \vdash e : \tau}{\lambda \vdash \ell := e : \tau\ \mathsf{cmd}} \qquad \frac{\lambda \vdash p_1 : \tau\ \mathsf{cmd} \qquad \lambda \vdash p_2 : \tau\ \mathsf{cmd}}{\lambda \vdash p_1; p_2 : \tau\ \mathsf{cmd}}$$

$$\frac{\lambda \vdash e : \tau \qquad \lambda \vdash p_1 : \tau\ \mathsf{cmd} \qquad \lambda \vdash p_2 : \tau\ \mathsf{cmd}}{\lambda \vdash \mathsf{if}\ e\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 : \tau\ \mathsf{cmd}} \qquad \frac{\lambda \vdash e : \tau \qquad \lambda \vdash p : \tau\ \mathsf{cmd}}{\lambda \vdash \mathsf{while}\ e\ \mathsf{do}\ p : \tau\ \mathsf{cmd}}$$

$$\frac{\lambda \vdash p : \tau\ \mathsf{cmd} \qquad \tau' \sqsubseteq \tau}{\lambda \vdash p : \tau'\ \mathsf{cmd}}$$