Homework SOS
Master Science Informatique 2018–2019

This homework is handed out on October 10th and is due November 4th 2018. Answers should be sent by mail to `Delphine.Demange@irisa.fr` *and* `Thomas.Jensen@irisa.fr`*. The deadline is strict. You can prepare the work in groups of one or two people (send one copy per group).*

We consider a small imperative language for programming a machine that has a fixed number of registers (ranged over by $\mathtt{x}, \mathtt{y}, \ldots \in \mathit{Var}$ and a memory whose addresses are positive integers and which contains integers. The memory can be read and written freely. The syntax of the language is given below.

Expressions

$$\begin{array}{rcll} \exp \ni e & ::= & x \mid n \mid e_1\ o\ e_2 & o \in \{+, -, \times, =, \leq\} \\ & & \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{not}\ e \mid e_1\ \mathtt{and}\ e_2 \mid e_1\ \mathtt{or}\ e_2 & \\ & & \mid [e] & \end{array}$$

Commands

$$\begin{array}{rcl} \mathit{Comm} \ni S & ::= & \mathtt{x} := e \mid [e_1] := e_2 \mid \mathtt{skip} \mid S_1; S_2 \\ & & \mid \mathtt{if}\ e\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2 \mid \mathtt{while}\ e\ \mathtt{do}\ S \end{array}$$

$n \in \mathit{Num},\ \mathtt{x} \in \mathit{Var}$

Example:
$\mathtt{x} := 100;$
$[\mathtt{x}] := 42;$
$\mathtt{y} := \mathtt{x};$
$\mathtt{i} := 1;$
$\mathtt{while}\ (i \leq 10)\ \mathtt{do}\ ([x + i] := [y + i - 1];\ i := i + 1)$

Here, the variables `x` and `y` are used as references (pointers) to the memory. Because of aliasing, the example program ends with the memory segment from 100 to 110 equal to $[42, 42, 42, 42, 42, 42, 42, 42, 42, 42]$.

Question 1. Give a small-step semantics to the language:

- Define a **State** domain for this machine.
- Define a denotational semantics of expressions.
- Define a structural operational semantics (small-step) of commands.

We expect to see a short but precise discussion of the choices that you make when defining the new domain and semantics. In particular, discuss the way memory is modelled, and how errors are modeled.

Question 2. In this language it is possible to make programming mistakes, due to the fact that it is easy to mix integers, booleans and memory references, and because the language permits arbitrary computations with memory references.

Define a type system for the language that helps the programmers separate integers, booleans and memory references. We suppose that every variable is given one and only one type (either reference, boolean or integer) throughout the whole program. We write $\Gamma(x)$ for the type of a variable x.

- Give the precise definition of types you consider in your type system.
- The type system should restrict the way that references can be manipulated by the program, but still leave the possibility to make "reasonable" pointer arithmetic. Describe more precisely what you consider "reasonable" and define what it means to be well-typed and which properties your system is intended to guarantee.
- Define the type judgment $\Gamma \vdash S$ ("command S is well-typed in the typing context Γ"), by giving the corresponding typing rules, and explain why your rules enforce the guarantees you previously chose.

Question 3. The type system is not strong enough to exclude certain runtime errors (such as division by 0 or trying to access a bad memory address) statically. Define a static analysis of well-typed programs for tracking values of expressions in the language, with the goal of avoiding such runtime errors.

- Identify and describe a set of runtime errors to be detected.
- Define an abstract domain for detecting such errors.
- Write a static analysis for the language based on your abstract domain.
- Give examples showing what errors the analysis is capable of detecting.