Prof. Dr. Peter Thiemann
Matthias Keil Summer Term 2015

# Essentials of Programming Languages

http://proglang.informatik.uni-freiburg.de/teaching/konzepte/2015/

## Exercise Sheet 11

### 11.1 Big-step Operational Semantics

As the last exercise sheets were considered to small-step operational semantics, this exercise sheets now considers the big-step semantics of our mini language.

A (big-step) operational semantics describes the behavior of a programming language by specifying an abstract machine for it. As inputs, it uses the expressions of the languages and a state is formalized by an expression, current program heap, and the current environment, if necessary. We specify a transition function for each state and thus define the behavior of the abstract machine. When we formalize it, our transition relation is represented by a judgment of the form

$$\rho \vdash \langle \sigma, e \rangle \Downarrow \langle \sigma', v \rangle$$

where $\rho$ represents our current environment, $\sigma$ the current program heap, $e$ is an arbitrary expression, and $v$ is a value.

Let's take the small arithmetic language

$$e ::= n \mid e_1 + e_2$$

we had at the beginning of the lecture. Here, $n$ is an integer number and represents the only value possible. So, if we get an integer constant $n$ we don't do anything, as it is already a number. Thus, our transition rule looks like the following:

$$\rho \vdash \langle \sigma, n \rangle \Downarrow \langle \sigma, n \rangle$$

For the other type of expression, $e_1 + e_2$, we had some preconditions in our interpreter: Both $e_1$ and $e_2$ had to be evaluated to a value, before we could do something. We would write such a precondition and conclusion like this:

$$\frac{Precondition}{Conclusion}$$

Thus, the transition function for $e_1 + e_2$ is

$$\frac{\rho \vdash \langle \sigma, e_1 \rangle \Downarrow \langle \sigma, v_1 \rangle \qquad \rho \vdash \langle \sigma, e_2 \rangle \Downarrow \langle \sigma, v_2 \rangle}{\rho \vdash \langle \sigma, e_1 + e_2 \rangle \Downarrow \langle \sigma, v_1 \oplus v_2 \rangle}$$

which could be read as "if, under environment $\rho$, $\langle\sigma, e_1\rangle$ evaluates to $\langle\sigma, v_1\rangle$ (and similar for $\langle\sigma, e_2\rangle$), then, under environment $\rho$, $\langle\sigma, e_1 + e_2\rangle$ evaluates to $\langle\sigma, v_1 \oplus v_2\rangle$". Here, $\oplus$ represents the addition function of our meta-language.
Write all the transition rules for a big-step operational semantics for the language

$$e ::= n \mid x \mid \texttt{op}(e, e) \mid \lambda x.e \mid e(e) \mid \texttt{new} \mid e.x \mid e.x := e$$

where

$$v ::= n \mid \ l$$
$$o ::= \emptyset \mid \ o[x \mapsto v]$$
$$s ::= o \mid \ (\rho, \lambda x.e)$$

in the formalism above. You should use closures instead of substitutions (as introduced in the lecture). Values are numbers or locations. The program heap maps locations to storables s, which are either closures $(\rho, \lambda x.e)$ or objects o. The `put` method for an environment (heap) is formalized as $\rho[x \mapsto v]$ $(\sigma[l \mapsto s])$ and the `get` method is formalized as $\rho(x) = v$ $(\sigma(l) = s)$.

## Submission

**Deadline** The submission deadline is **24.07.2015, 12:00 (noon)**. Late submissions will not be accepted. Submit your solution to the subversion repository.

**Submission** Your solution will consist of one *folder* (**exercise11**) for each exercise sheet. Submit one *pdf* file (**⟨name⟩_exercise11_⟨nr⟩.pdf**) and one *rkt* file (**⟨name⟩_exercise11_⟨nr⟩.rkt**) per exercise.

Your solution may be either in German or in English. Clear and understandable style is required. You are strongly encouraged to test your solution. Your code must compile without errors (which did not necessarily mean that everything has to work). Provide your source code with comments to understand the intention.