ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 10: Big Step Semantics

Submission deadline: May 7th, 2012

Please submit your solution before **10:15am** on the submission date specified above. Solutions can be submitted via e-mail or by using the boxes to the left of **CAB F 51.1**. Make sure that the first page (and preferably each sheet) always contains your name, the exercise sheet number as well as your tutor's name and the weekday (Tuesday or Wednesday) of your exercise group. Don't forget to staple your pages if you submit more than one page.

Assignment 1

Consider the following **IMP** statement s:

```
while n # 0 do
    a := a+n;
    b := b*n;
    n := n-1
end
```

Let σ be a state such that $\sigma(\mathtt{a}) = 0$, $\sigma(\mathtt{b}) = 1$, and $\sigma(\mathtt{n}) = 2$. Prove using the natural semantics that there is a state σ' with $\sigma'(\mathtt{a}) = 3$, $\sigma'(\mathtt{b}) = 2$, and $\sigma'(\mathtt{n}) = 0$ such that $\langle s, \sigma\rangle \to \sigma'$.

Provide the complete derivation tree. You have to explicitly write the names of the rules you apply at each derivation step.

Assignment 2

In the lecture, you have seen the proof of the direction from left to right of the following claim:

$\langle \mathtt{while}\ b\ \mathtt{do}\ s\ \mathtt{end}, \sigma\rangle \to \sigma' \Leftrightarrow \langle \mathtt{if}\ b\ \mathtt{then}\ s; \mathtt{while}\ b\ \mathtt{do}\ s\ \mathtt{end}\ \mathtt{else}\ \mathtt{skip}\ \mathtt{end}, \sigma\rangle \to \sigma'$

Prove the direction from right to left of the claim.

Assignment 3

Consider the extension of the **IMP** programming language with a `for` statement:

$$\texttt{for } x := e_1 \texttt{ to } e_2 \texttt{ do } s \texttt{ end}$$

The execution of the statement first evaluates e_1 and assigns the result to variable x. Then, while the value of x is not equal to that of e_2, it executes s and increases x by one.

There are two possible interpretations of the above statement: (i) e_2 is evaluated once, (ii) e_2 is evaluated before each comparison to the value of x.

(a) Provide derivation rules in natural semantics for both (i) and (ii). Do *not* use `while` in your derivation rules.

(b) Consider the case of semantics (ii) only. Show that for any $x, e_1, e_2, s, \sigma, \sigma'$:

$$\langle \texttt{for } x := e_1 \texttt{ to } e_2 \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma' \;\Rightarrow\; \langle x := e_1;\ \texttt{while } x\#e_2 \texttt{ do } s; x := x + 1 \texttt{ end }, \sigma\rangle \to \sigma'$$

Assignment 4

In this assignment you will write a simple interpreter for **IMP** programs. You will use the programming language Haskell. A skeleton of the **IMP** interpreter as a literate Haskell file is available at the course web page. The skeleton file contains the data types for arithmetic expressions, Boolean expressions, and statements for representing **IMP** programs in Haskell. Moreover, the skeleton file contains some auxiliary functions. You have to use this skeleton file instead of what you have developed in assignment 4 of exercise sheet 9. The skeleton contains already a parser and several auxiliary functions (e.g., evaluation of arithmetic and Boolean expressions).

Download the skeleton file and complete the definition of the function

```
transNS :: Config -> Config
```

The place where you should insert your code in the skeleton file is marked by the word `TODO`. The function `transNS` should encode the rules of the transition relation from the lecture for the natural semantics. Feel free to extend **IMP**, e.g., with local variables.

Please mail your solution of this assignment together with some test cases to your tutor. The email addresses of the tutors are:

Malte Schwerhoff	`malte.schwerhoff@inf.ethz.ch`
Yannis Kassios	`ioannis.kassios@inf.ethz.ch`
Alex Summers	`alexander.summers@inf.ethz.ch`

Assignment 5 - Headache of the week

Extend the natural semantics of **IMP** to support a `break` statement. The statement should stop the execution of the innermost loop that is being executed, as in C. When executing a `break` statement, the control should go immediately after the loop whose execution is being broken. You may assume that `break` will never appear outside of a loop.