ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 9: Small Step Semantics

Submission deadline: May 5th, 2009

Assignment 1

Consider again the statement s from Assignment 2 on the previous exercise sheet:

```
y := 0;
while x>0 do
  y := y + x;
  x := x - 2
end
```

Prove that there is a state σ' with $\sigma'(\mathtt{y}) = 4$ such that $\langle s, \sigma\rangle \rightarrow_1^* \sigma'$, where σ is a state with $\sigma(\mathtt{x}) = 3$. Compare the derivation sequence with the derivation tree from the Assignment 2(b) on the previous exercise sheet.

Assignment 2

Assume that $\langle s_1; s_2, \sigma\rangle \rightarrow_1^* \langle s_2, \sigma'\rangle$. Show that it is not necessarily the case that $\langle s_1, \sigma\rangle \rightarrow_1^* \sigma'$.

Assignment 3

Let s_1 and s_2 be statements, σ, σ' states, and k a positive integer.

Prove that if $\langle s_1, \sigma\rangle \rightarrow_1^k \sigma'$ then $\langle s_1; s_2, \sigma\rangle \rightarrow_1^k \langle s_2, \sigma'\rangle$. Intuitively speaking, this means that the execution of s_1 is not influenced by the statement that follows s_1.

Assignment 4

Extend **IMP** programming language with the statement

$$\texttt{assert } b \texttt{ before } s$$

where b is a Boolean expression and s a statement. The intuitive meaning is that if b evaluates to true then we execute s and otherwise the execution of the complete program aborts. Extend the structural operational semantics to capture this (do not use an `abort` statement).

Show that `assert` $1 = 1$ `before` s is semantically equivalent to s but that `assert` $0 = 1$ `before` s neither is semantically equivalent to `while` $1=1$ `do skip end` nor to `skip`. What changes when you extend the natural semantics to capture the semantics of the new statement?

Assignment 5

In the lecture, you have seen an extension of the **IMP** programming language with an operator `par` for expressing parallel computations. You have also seen the SOS rules for an interleaved execution of the statement s_1 `par` s_2.

In the following, let s and s' be statements that are semantically equivalent.

(a) Give an example that shows that in general $(s_1; s; s_2)$ `par` t is not semantically equivalent to $(s_1; s'; s_2)$ `par` t.

(b) Prove that under the additional assumption that t is a statement with $FV(s) \cap FV(t) = \emptyset$ and $FV(s') \cap FV(t) = \emptyset$, we have that

$$(s_1; s; s_2) \texttt{ par } t \qquad \text{and} \qquad (s_1; s'; s_2) \texttt{ par } t$$

are semantically equivalent.

For proving the semantic equivalence, you might first prove some auxiliary lemmas, e.g., for statements u, v, and r, we have that $(u; v)$ `par` r is semantically equivalent to $u; (v$ `par` $r)$ if $FV(u) \cap FV(r) = \emptyset$. Moreover, convince yourself that the execution of a statement r only depends on the values of the variables that occur in r and only these values can change in an execution step of r.

Assignment 6

In this assignment you will extend the simple **IMP** interpreter with the structural operational semantics. Download the skeleton file from the course web page and implement the function

```
transSOS :: Config -> Config
```

that encodes the rules presented in the lecture for the structural operational semantics. The place where you have to insert your code in the skeleton file are marked by `TODO`.

Compare your implementation of `transSOS` with the function `transNS` that implements the rules for the natural semantics.