ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 10: Small Step Semantics

Submission deadline: May 17th, 2010

Please submit your solution before **9:15am** on the submission date specified above. Solutions can be submitted via e-mail or by using the boxes to the left of **RZ F1**. Make sure that the first page 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

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 2

Let s_1 and s_2 be statements, σ and σ' states, and k a positive integer. Prove that if $\langle s_1, \sigma\rangle \rightarrow_1^k \sigma'$ then $\langle s_1; s_2, \sigma\rangle \rightarrow_1^k \langle s_2, \sigma'\rangle$.

Assignment 3

In this assignment you will extend the simple IMP interpreter with the structural operational semantics. Download the skeleton file `simpi_skeleton2.lhs` 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.

Please mail your solution of this assignment to your tutor. The email addresses of the tutors are:

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

Assignment 4 - Headache of the week

Coroutines are parallel programs that handle their own scheduling. Only one coroutine is executing at any given time. We call that coroutine the *active* coroutine. The active coroutine may *yield* control to another coroutine, which then becomes the active coroutine, and so forth. When a coroutine becomes active, it resumes execution exactly at the point where it was executing when it was last active.

In this task, we implement a limited form of coroutines. Let s_1, s_2 be statements. The following construct turns them into coroutines:

$$s_1 \texttt{ copar } s_2$$

When s_1 `copar` s_2 executes, the active coroutine is initially s_1.

The `yield` statement yields the control. In particular, when s_1 executes `yield`, then s_2 becomes active and when s_2 executes `yield` then s_1 becomes active.

When a `yield` is executed, the newly activated coroutine resumes execution immediately after the last yield that it executed. If no such point exists (i.e. s_1 executes `yield` for the first time), then the control goes to the beginning of newly activated coroutine.

Once one of the coroutines terminates, then the whole construct terminates. Exceptionally, if the last statement that one coroutine executes is a `yield`, then the construct does not yet terminate, but yields the control to the other coroutine.

Consider for example the following program:

```
x:=0;y:=0;
(while x<10 do x:=x+1; yield end) copar (while 1=1 do y:=y+x; yield end)
```

After initializing the variables `x,y`, the program introduces two coroutines. The left coroutine starts executing first. The left coroutine runs a loop on `x`. Until `x` is equal to 10, the left coroutine increments it and then yields the control to the right coroutine. The right coroutine adds `x` to `y` and then yields the control back to the left coroutine.

The `copar` statement executes until the left coroutine terminates. At that point, `x` equals 10, while `y` equals $\sum_{i=1}^{10} i$ (which, incidentally, is 55).

For simplicity, we do not allow nested coroutines, i.e. the following would be illegal:

$$(s_1 \texttt{ copar } s_2) \texttt{ copar } s_3$$

Extend the structural semantics of **IMP** to support coroutines as described above.