Generic Language Technology (2011-2012)
Assignment 3 (deadline: December **12** 2012)
Link to Formal Verification of DSL Models - Dynamic Semantics
Additional instructions

Introduction

Some students asked for additional instructions regarding the assignment. Here are some clarifications and hints which may help students to accomplish this assignment.

Further explanation regarding the syntax

A program/model in the `SSM` language is any SimpleProcess. For instance,
`send` $sign1 \parallel_{\{sign1\}}$ `receive` $sign1$ or
`send` $sign1 \parallel_{\{sign1\}}$ `receive` $sign2$ or
`send` $sign1 \parallel$ `receive` $sign2$ or
$skip;$`receive` $sign$ or
$skip;$`receive` $sign \parallel$ `send` $sign1$

When it comes to concrete syntax of the language, you are given freedom to define it, for instance to introduce brackets, priorities,... If after this you still encounter problems with executing an arbitrary model in the language you are allowed to make some (reasonable) restrictions on the input terms (for instance, you are allowed to restrict to terms containing not more than one occurrence of the constructs $\parallel_S$ and $\parallel$ and, if it appears, it is the outermost construct of the term).

You do not need to define a typechecker.

Further explanation of the semantics

The phrase "a given set of signal names" should be understood that the set of signals used in the model is predefined, if that is needed in the tool you are using. For some tools this is required. The same holds for "a given set of variables".

The semantics you define should correspond to the explanation given in the assignment. You can decide in which format the execution of an input model is represented. (Hint: you may represent it as a list of sequences of events, for instance:
input term: `send` $sign1 \parallel_{\{sign1\}}$ `receive` $sign1$
output list: $[sign_1]$

input term: `send` $sign1 \parallel_{\{sign1\}}$ `receive` $sign2$
output list: $[deadlock]$

input term: `send` $sign1 \parallel$ `receive` $sign2$
output list: [[`send` $sign1$, `receive` $sign2$], [`receive` $sign2$, `send` $sign1$]].)
Note that, for instance, the (correct) execution of $skip$; `receive` $sign \parallel$ `send` $sign1$ depends on the concrete syntax (priorities) you decide to define.

Many students observed that the structural operational semantics (SOS) is a well fitting formalism for defining the semantics of the `SSM` language. A slight difference with the examples we have seen at the lectures is that you may need to have labelled transitions now. You should think what these labels should denote.

Describing semantics in ASF+SDF

If you have chosen to work with ASF+SDF, you should consider using equations, which describe how a term should be rewritten. You can look at an example (at `http://www.win.tue.nl/~andova/education/GLT/ASFSDFexample/`) of an LTS language with an .asf file containing equations examples. By these equations certain functions on LTSs are defined and can be applied and corresponding results are returned.