The PRISM Language - Semantics

This document gives the semantics for the PRISM language. For details of the language itself, see the manual:

- http://www.cs.bham.ac.uk/~dxp/prism/manual/ThePRISMLanguage

An important observation is that we do not define the semantics in a compositional manner, i.e. by first giving the semantics of each module in the system and then combining these results. The reason for this is that guards (and updates) of one module are allowed to refer to the variables of other modules (and indeed global variables). Instead, we define the semantics of a system by translating its set of modules into a single "system module" (in a compositional manner) and then defining the semantics for the system through this single module.

Constructing the system module

In this section, we describe the process of constructing the system module from its component modules. The composition of the modules is defined by a process-algebraic expression which can include parallel composition of modules, action hiding and action renaming. We now consider each of these in turn. Note that, in this construction process, we require that all updates of all commands have been expanded to *explicitly* include all local variables of the module and all global variables, even those that do not change.

Parallel composition

Although there are three types of parallel composition, we need only consider the case $M_1|[A]|M_2$, since $M_1|||M_2$ is equivalent to $M_1|[\emptyset]|M_2$ and $M_1||M_2$ is equivalent to $M_1|[A_1 \cap A_2]|M_2$ where A_i is the set of actions that appear in module M_i. The commands of the module $M = M1|[A]|M2$ are constructed according to the follow rules:

1. for each command $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of $M1$,
   add $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;

2. for each command $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of $M2$,
   add $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;

3. for each $a \notin A$ and command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of $M1$,
   add $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;

4. for each $a \notin A$ and command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of $M2$,
   add $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;

5. for each $a \in A$, command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of $M1$ and command $[a]\ g' \rightarrow \lambda'_1 : u'_1 + \cdots + \lambda'_{n'} : u'_{n'}$ of $M2$,
add

$$\begin{array}{rlllll} [a]\ g \,\&\, g' \rightarrow & \lambda_1 * \lambda'_1 : u_1 \,\&\, u'_1 & + \cdots + & \lambda_n * \lambda'_1 : u_n \,\&\, u'_1 \\ & +\lambda_1 * \lambda'_2 : u_1 \,\&\, u'_2 & + \cdots + & \lambda_n * \lambda'_2 : u_n \,\&\, u'_2 \\ & & \vdots & \\ & +\lambda_1 * \lambda'_{n'} : u_1 \,\&\, u'_{n'} & + \cdots + & \lambda_n * \lambda'_{n'} : u_n \,\&\, u'_{n'} \end{array}$$

to the commands of M.

Action hiding

The commands of $M = M'/A$ are constructed according to the follow rules:

1. for each command $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;
2. for each $a \notin A$ and command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;
3. for each $a \in A$ and command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M.

Action renaming

The commands of $M = M'\{a_1 \leftarrow b_1, \ldots, a_m \leftarrow b_m\}$ are constructed as follows:

1. for each command $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;
2. for each $a \notin \{a_1, \ldots, a_m\}$ and command $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[a]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M;
3. for each $1 \leq i \leq m$ and command $[a_i]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ of M',
add $[b_i]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$ to the commands of M.

The semantics of the system module

In this section, we give the semantics of a system module, as constructed through the rules in the previous section. We suppose that:

- C is the multiset of commands generated by the rules above;
- $V = \{v_1, \ldots, v_m\}$ is the set of variables, both local and global, that appear in the system description.

Regardless of model type (DTMC, MDP or CTMC), we construct the state space of the system as follows. A state is a tuple $(x_1, \ldots, x_m)$ where x_i is a value for the variable v_i. The set of all states S is therefore the set of all possible valuations of the variables in V. The set of initial states can be specified in one of two ways: either by giving an initial value for each variable, or by giving a predicate over variables (using the `init...endinit` construct). In

the former case, $\bar{S} = \{\bar{s}\}$ where $\bar{s} = (\bar{x}_1, \ldots, \bar{x}_m)$ and $\bar{x}_i$ is a the initial value of the variable v_i (recall that, if the initial value of a variable is left unspecified, it is taken to be the minimum value of the variable's range). In the latter case, $\bar{S}$ is the subset of states S which satisfy the predicate specified in the `init...endinit` construct.
We now consider the semantics for a single command of the system module. From this point, we ignore any action-labels assigned to commands in C; these were required only for the process-algebraic construction and can now be safely discarded. Hence, each command c of C takes the form:

$$[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$$

Since the guard g is a predicate over the variables in V and each state of the system is a valuation of these variables, g defines a subset of the global state space S. We denote this set of states $S_c = \{s \in S \mid s \models g\}$.
Each update u_j of c corresponds to a transition that the system can make when in a state $s \in S_c$. The transition is defined by giving the new value of each variable as an expression. Hence, we can think of u_j as a function from S_c to S. If u_j is $(v'_1 = expr_1) \wedge \cdots \wedge (v'_m = expr_m)$, then for each state $s \in S_c$:

$$u_j(s) = (expr_1(s), \ldots, expr_m(s))$$

Using the value λ_j associated with each update u_j, the command c defines, for each $s \in S_c$, a function $\mu_{c,s} : S \rightarrow \mathbb{R}_{\geq 0}$ where for each $t \in S$:

$$\mu_{c,s}(t) \stackrel{\text{def}}{=} \sum_{\substack{1 \leq j \leq n \\ \wedge u_j(s) = t}} \lambda_j$$

Note that, for DTMCs and MDPs, the syntactic constraints placed on the constants λ_j mean that the function $\mu_{c,s}$ is actually a probability distribution over S.
Finally, we can now define the probabilistic model itself, i.e. the probability matrix $\mathbf{P}$ for a DTMC, the function *Steps* for an MDP, or the transition rate matrix $\mathbf{R}$ for a CTMC.

DTMC semantics

We define the transition probability matrix $\mathbf{P} : S \times S \rightarrow [0, 1]$ to be the matrix $\bar{\mathbf{P}}$ after its rows have been normalised where for any $s, t \in S$:

$$\bar{\mathbf{P}}(s, t) = \sum_{c \in C} \mu_{c,s}(t)\,.$$

The normalisation is required since the values appearing in any row of this matrix $\bar{\mathbf{P}}$ can sum to more than one, through either the nondeterminism introduced through the parallel composition of modules or local nondeterminism in a module (i.e. overlapping guards). This normalisation can be considered as replacing any nondeterministic choice between a set of transitions with a uniform (probabilistic) choice between the transitions.

MDP Semantics

If the model is an MDP, the function $Steps : S \rightarrow 2^{Dist(S)}$ is such that for any $s \in S$:

$$Steps(s) = \{\mu_{c,s} \mid c \in C \text{ and } s \in S_c\}\,.$$

CTMC Semantics

For a CTMC, the transition rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is such that for any $s, t \in S$:

$$\mathbf{R}(s,t) = \sum_{c \in C} \mu_{c,s}(t)\,.$$