PRISM

This is a quick overview of the PRISM probabilistic model-checker.

- It focuses mostly on the PRISM language for specifying models.
- We will meet methods for specify model *properties* later.
- In the lab you will be using the PRISM software to run model *simulations* and *verify* model *properties*.

More detail on all this material can be found in the PRISM manual:

- On-line at `http://www.prismmodelchecker.org/manual/`
- A local copy is in the PRISM installation directory (usually `/usr/prism/`) in `doc/`. There is a single PDF version here and (in `doc/manual/`) an HTML version like the on-line manual.

The manual is quite short; you are recommended to read pp 10-22 now and pp 23 -40 soon! (Page numbers refer to the PDF version.)

Types of Probabilistic Automaton

We have seen an formal example and a formal definition of the *discrete-time Markov chain*

- Time proceeds in discrete ticks; any choice of action from a state is resolved probilistically: actions are labelled with probabilities.

Other types include

- The *continous-time Markov chain* (CTMC)
  - Elapsed time is a positive real number, not an integer,
  - actions are labelled with a *rate*, λ rather than a probability.
  - The probability of the action is occuring later than time t after arriving at the source state is $e^{-\lambda t}$ (an *exponential disribution*).
  - So, when there are several actions from a state there is a *race condition* between them.
- The *Markov decision process* (MDP)
  - Time is discrete (an integer), like a DTMC
  - There is *indeterminism* as well as probability: from a state, the automaton can make a non-deterministic choice between probability distributions, then a probabilistic (random) choice between actions *within* the chosen probability distribution.

Types of Probabilistic Automaton

- The *Probabilisitc Timed Automaton* (PTA)
  - A *timed automaton* as used in (eg) UPPAAL
  - There are continuous-time *clocks* which can be *reset* when an action is taken, as well as probabilistic or non-deterministic choice of actions from a state.

The theory of CTMCs, MDPs, and PTAs are beyond the scope of the present short lecture series. CTMCs especially are mathemtically rather sophisticated as then involve uncountably infinite probability spaces.

All four types are supported by PRISM: we will mainly be using DTMCs but all are easy to use in PRISM. This support is discussed in the manual, pp 11-21.

PRISM - Model Syntax

Last week we saw a PRISM model of a process simulating a die-roll with coin tosses. The manual starts with this model of two identical processes operating with mutual exclusion:

```
mdp
module M1
  x : [0..2] init 0;

  [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
  [] x=1 & y!=2 -> (x'=2);
  [] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);
endmodule

module M2
  y : [0..2] init 0;

  [] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
  [] y=1 & x!=2 -> (y'=2);
  [] y=2 -> 0.5:(y'=2) + 0.5:(y'=0);
endmodule
```

PRISM - Model Syntax

Structure of a PRISM model

- a declaration of a model type (DTMC, CTMC, MDP or PTA).
- followed by one or more *modules* and optional stuff such as a *rewards* declaration (see later)
- The die-roll simulation had only a single module but this one has two: M1, M2: two processes running in parallel.
- PRISM generates from the model a probabilistic automaton or appropriate type. This will be a product or *parallel composition* of automata generated by the modules.

PRISM - Modules

A Module is defined by a list of declarations of *variables* followed by a list of *commands*.

- (Local) variables define the module *state* - See manual, p 12
  - M1 declares x, an integer in range 0..2, initially 0
  - M2 declares y, an integer in range 0..2, initially 0
  - The mutually exclusive *critical sections* are respectively the states, x=2, y=2
- Commands define the module behaviour - See manual, p 13
  - Format of a command is `[]` *guard* `->` *updates* `;`
  - The a guard is a boolean expression composed of predicates on the variable values
    - Defines a set of states
    - NB `&`, `|`
  - The updates have format $p_1 : \mathtt{upd}_1 + \ldots p_n : \mathtt{upd}_n$
    - $p_1 \ldots p_n$ are probabilities which must add up to 1
    - Each $\mathtt{upd}_k$ is a list of one or more clauses conjoined with &, each with format
    - $(v' = \mathit{expression})$ - v is a variable; the expression is made of constants and unprimed variables.

PRISM - more on Commands

Examples

- `[] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);`
  - If we are in state x=0 then with probability 0.8, update x to 0 and with probability 0.2, update x to 1.
- `[] x=1 & y!=2 -> (x'=2);`
  - If we are in state x=1 & y != 2 then (unconditionally) update x to 2.
- `[] x1=0 & x2>0 & x2<10 -> 0.5:(x1'=1)&(x2'=x2+1) + 0.5:(x1'=2)&(x2'=x2-1);`
  - If we are in a state with $x_1 = 0$ and $x_2 > 0$ and $x_2 < 10$ then ...
  - with probability 0.5: update x_1 to 1 and x_2 to (old) $x_2 + 1$ and ...
  - with probability 0.5: update x_1 to 2 and x_2 to (old) $x_2 - 1$.

The [] beginning a command may optionally contain a *label*: this will be useful for *synchronising* commands (see below)

The automaton for a model is the *parallel composition* of the automata for the modules: state is made up of all the state variables. If more than one guard is satisfies there is a choice of actions: nondeterministic in case of MDP and proabilistic in case of DTMC.

A Second example from the manual

```
// N-place queue + server
ctmc
const int N = 10;
const double mu = 1/10;
const double lambda = 1/2;
const double gamma = 1/3;

module queue
     q : [0..N];

     [] q<N -> mu:(q'=q+1);
     [] q=N -> mu:(q'=q);
     [serve] q>0 -> lambda:(q'=q-1);
endmodule

module server
     s : [0..1];

     [serve] s=0 -> 1:(s'=1);
     [] s=1 -> gamma:(s'=0);
endmodule
```

Second example

- A CTMC - manual p 15; updates labelled with *rates* rather than probabilities;
  - As with DTMC probabilities, several updates can be combined with '+' but the constraint that probabilities add up to 1 is gone.
- Notice declaration of *constants*
- Read the manual (pp 16, 17) on constants, *expressions* and *built-in functions*: you will notice a lot of similarity with C programming.
- Two of the commands are labelled `[serve]`
  - This forces *synchronisation*: manual p 17
  - A transition of the model (composite) automaton involving one of these commands fire both of them simultaneously.
  - Common in parallel composition of automata.
  - More than two commands can be synchronised.

Module Renaming

The first example (slide 4) has M2 specified in full: but it is in fact the same as M1 but with the roles of x, y, reversed.
Here is the first example again with *renaming* to save some writing (and potential for mistakes) ...

```
mdp
module M1
  x : [0..2] init 0;

  [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
  [] x=1 & y!=2 -> (x'=2);
  [] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);
endmodule

module M2 = M1 [ x=y, y=x ] endmodule
```

Costs and Rewards

- In a model of a process, you may be interested in the time (or the number of steps) to reach a particular state.
  - How long does it take
  - How many retries
  - How many irrecoverable errors
  - etc
- In a probabilistic model, this will be the *expected* time/number of steps

A “reward” is a numerical value that can be attached to (some or all) states, or (some or all) transitions.

- It might be a “cost” rather than a “reward” – depending on whether we think it good or bad.
- It is accumulated in runs of the model
  - or an expected (average over all possible runs) value computed from the probability structure

Costs and Rewards

See manual, pp 21-22

- Example: the dice-throw simulation (Introductory lecture) could have alongside the module defnitions,

```
rewards
true : 1;
endrewards
```

- This literally counts all states visited before the model gets to one of the terminal states (with the self-loops).
- As seen in the introductory lecture and also in the PRISM tutorial Part 1, the average (expected) value is $3\frac{2}{3}$

Costs and Rewards

Another example:

```
rewards
    x=0 : 100;
    x>0 & x<10 : 2*x;
    x=10 : 100;
endrewards
```

- Multiple reward items. For every guard satisfied, the corresponding reward is applied; the total reward is the sum of the rewards applied (although the guards are mutually exclusive here).
- A reward can be a function of the state variables.

Costs and Rewards

A reward structure giving rewards to actions (transitions):

```
rewards
    [] true : 1;
    [a] true : x;
    [b] true : 2*x;
endrewards
```

- State-based and action-based reward definitions can go in the same structure.

There can be multiple reward structures in a model, and they can be labelled:

```
rewards "total_time"
    true : 1;
endrewards]

rewards "num_failures"
    [fail] true : 1;
endrewards
```

What next?

You need to practise using PRISM

- Review your work through PRISM tutorial Part 1 (`http://www.prismmodelchecker.org/tutorial/`)
- Look at / do more of these - at least Parts 2, 3, 5, 6

Read ahead

- Property specification manual pp 23-40 covers how *properties* of a model can be set up to be checked (verified). This includes *reachability* properties which we cover next, and rewards.
- Running PRISM manual pp 40-64 - mostly reference.