02246 Mandatory Assignment
PRISM and Probabilities

Flemming Nielson, Michael Smith, Lijun Zhang

Fall 2013

1 Adding Probabilities to PRISM

So far in PRISM, we've been dealing with purely discrete models, in the sense that they contain non-determinism, but do not carry any *quantitative* information. In Part II of the course we will look at adding quantitative information to our models:

- **Part II**: what's the *probability* that we choose a particular transition out of a state?

For the purposes of Part II of the course, we're going to be looking at *purely probabilistic* models, which means that the models do not contain any non-determinism — whenever a state in the model has multiple outgoing transitions, we must assign a probability to each[1]. Mathematically, this corresponds to a *discrete time Markov chain (DTMC)*.

In PRISM, moving from transition systems — which you've been constructing in Part I of the course — to DTMCs is very easy! There are only three things you need to do differently, when writing a model:

1. You need to use the keyword "`dtmc`" at the start of the file, rather than "`mdp`". This is *very important* to remember — if you don't do this, then PRISM will not interpret your model in the way you expect, and you may get incorrect results when you try to model check it.

2. You are allowed to use *probabilistic* commands. Previously, for transition systems, you've written commands of the form:

$$[\langle ACTION \rangle] \;\; \langle GUARD \rangle \;\; \rightarrow \;\; \langle UPDATE \rangle \;;$$

   A probabilistic command allows you to choose between *different updates*, by assigning a probability to each of them:

$$[\langle ACTION \rangle] \;\; \langle GUARD \rangle \;\; \rightarrow \;\; p_1 : \langle UPDATE_1 \rangle + \cdots + p_n : \langle UPDATE_n \rangle \;;$$

   Note that the probabilities should all sum to one: $\sum_{i=1}^{n} p_i = 1$. If you want to choose only one update, with probability one, then the following are considered identical in PRISM:

$$[\langle ACTION \rangle] \;\; \langle GUARD \rangle \;\; \rightarrow \;\; \langle UPDATE \rangle \;;$$
$$[\langle ACTION \rangle] \;\; \langle GUARD \rangle \;\; \rightarrow \;\; 1.0 : \langle UPDATE \rangle \;;$$

[1] As an aside, it *is* possible to combine probabilistic and non-deterministic transitions, and indeed, PRISM supports this. If you do so, you end up with a type of model called a Markov decision process (MDP). The details of MDPs, and how to do model checking on them, are beyond the scope of this course — however, it may be the subject of an advanced course in January 2014.

3. You are *not allowed* to have any non-determinism inside a module. This means that you can't write two commands whose guards can *both* be true at the same time. If you try to do this, then PRISM will automatically resolve the non-determinism using a uniform distribution — i.e. if there are n possible commands, it will choose each with probability $\frac{1}{n}$. In general you should not rely on PRISM to do this for you, but instead specify the probabilities yourself — this way, you will be certain that the model is describing what you want it to! You can check for this by looking at the log (select the *Log* tab), after you build the model — PRISM will output a warning about overlapping commands, if it has to resolve any non-determinism for you.

Note that there are *no changes* to the way you define variables, or the way in which you compose modules.

A Note on Composition

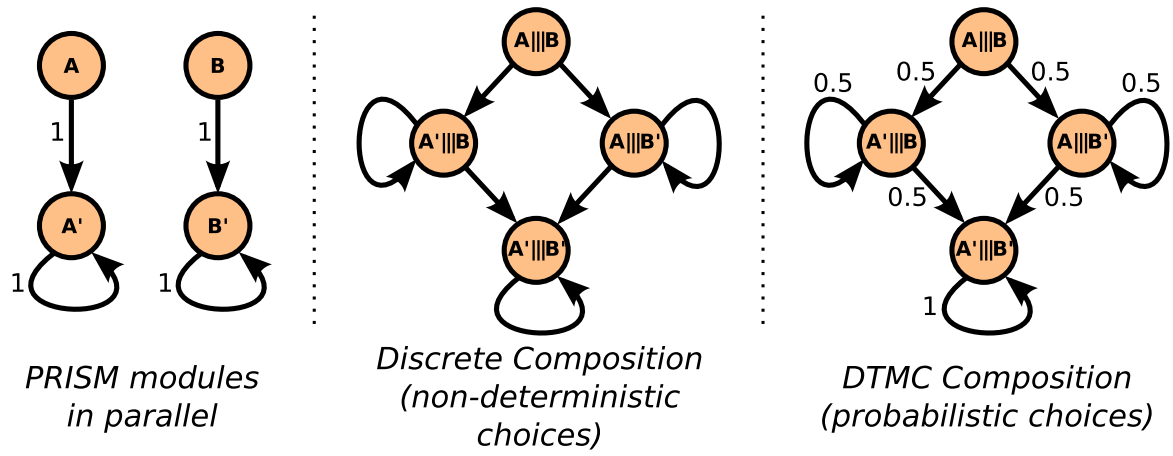

Figure 1: Independent composition of PRISM modules in a DTMC

For those of you who are paying attention, you'll notice at this stage that you've been given two pieces of conflicting information. You've just been told that DTMCs are purely probabilistic, and so you can't have any non-determinism when you write a PRISM module that describes a DTMC. However, we know from before that when you compose two modules that are independent (i.e. using the syntax $M_1|||M_2$), PRISM *interleaves* the execution of the modules — choosing the order of the interleaving non-deterministically.

In actual fact, when you describe a *DTMC* model in PRISM, these choices are *not* made non-deterministically, but instead an *equal probability* is given to each module executing. This is shown graphically in Figure 1 for two modules, each with two states. In the case of a transition system, as in Part I of the course, we introduce a non-deterministic choice between the transitions out of a state. In the case of a DTMC, however, there is an equal probability of choosing a transition from each of the modules.

2 Probabilistic Properties in PRISM

Once you've built a probabilistic model in PRISM, the next step is to analyse it. There are two ways in which you can approach this, in PRISM:

1. You can directly compute *transient* and *steady state* distributions of the DTMC. These features can be accessed from the main menu, under `Model > Compute`.

2. You can specify a property in *Probabilistic Computation Tree Logic (PCTL)*, and use the PRISM model checker to verify them. This is done using the *Properties* tab, in the same way as in Part I of the course.

For reference, here are some examples of how we write PCTL formulae in PRISM, where the metavariables Φ represent CTL state formulae. Atomic propositions can be specified in exactly the same way as in Part I of the course.

PCTL Formula	**PRISM Notation**
$\neg\Phi$	`!`Φ
$\Phi_1 \wedge \Phi_2$	Φ_1 `&` Φ_2
$\Phi_1 \Rightarrow \Phi_2$	Φ_1 `=>` Φ_2
$\mathbb{P}_{\leq p}(\Phi_1\ U\ \Phi_2)$	`P<=p[`Φ_1 `U` Φ_2`]`
$\mathbb{P}_{\geq p}(\Phi_1\ U^{\leq n}\ \Phi_2)$	`P>=p[`Φ_1 `U<=n` Φ_2`]`
$\mathbb{P}_{\leq p}(\bigcirc\ \Phi)$	`P<=p[X` Φ`]`
$\mathbb{P}_{\geq p}(\bigcirc\ \Phi)$	`P>=p[X` Φ`]`
$\mathbb{P}_{\leq p}(\Box\ \Phi)$	`P<=p[G` Φ`]`
$\mathbb{P}_{\geq p}(\Box^{\leq n}\ \Phi)$	`P>=p[G<=n` Φ`]`
$\mathbb{P}_{\leq p}(\Diamond^{\leq n}\ \Phi)$	`P<=p[F<=n` Φ`]`
$\mathbb{P}_{\geq p}(\Diamond\ \Phi)$	`P>=p[F` Φ`]`

Unlike with CTL, where we needed to encode the $\forall$ and $\exists$ quantifiers into the PRISM property specification format, the translation is more direct. This is because PRISM was designed to support PCTL model checking, rather than CTL model checking, as a primitive.

As a practical aside, PRISM also provides a useful extension to PCTL, which allows you to *ask for the probability* of a path formula holding, rather than only allowing you to compare it to some value. This can be written in PRISM as follows (where φ is a path formula):

`P=?[`φ`]`

When you use this notation, you have to be a little careful however, because its semantics is slightly different to 'normal', non-quantitative properties. Remember that PRISM by default will verify whether a property is true for *all initial* states in the model? Well, we can't do this for the above property unless there is a single initial state.

It is possible to query the probability of a path formula holding, starting in a different state, but unfortunately you need to use a different syntax for this. This is easiest to illustrate with a few examples:

PRISM Property	**Meaning**
`P>=1[G( P<=`p`[`φ`] )]`	Is it the case that for *all reachable states*, the probability of selecting a path that satisfies φ is $\leq p$?
`P>=1[G(` Φ `=> P<=`p`[`φ`] )]`	Is it the case that for *all reachable states satisfying* Φ, the probability of selecting a path that satisfies φ is $\leq p$?
`P=?[`φ`]`	What is the probability of selecting a path that satisfies φ, starting from *the initial state*?
`P=?[`φ `{`Φ`}]`	What is the probability of selecting a path that satisfies φ, starting from *the state that satisfies* Φ (if Φ holds of just one state)?
`P=?[`φ `{`Φ`} {max}]`	What is the *maximum* probability of selecting a path that satisfies φ, starting from *states that satisfy* Φ?
`P=?[`φ `{`Φ`} {min}]`	What is the *minimum* probability of selecting a path that satisfies φ, starting from *states that satisfy* Φ?

Note that the following pairs of properties in PRISM are equivalent, where $p \in [0, 1]$ is a probability

`P=?[`φ `{`Φ`} {min}] >=` p $\equiv$ `P>=1[G(` Φ `=> P>=`p`[`φ`] )]`
`P=?[`φ `{`Φ`} {max}] <=` p $\equiv$ `P>=1[G(` Φ `=> P<=`p`[`φ`] )]`

provided that all states are reachable.

3 Example: Simulating a Fair Die

To get started with using PRISM for probabilistic modelling and verification, let's take a look at an example model. In the lectures, we saw how we could simulate a fair die, using an unbiased coin — we saw how to write this directly as a DTMC, but let's encode it in PRISM as follows. You can download the model `dice.pm` from the *Mandatory Assignment Materials* directory on CampusNet.

```
module Die

  // Local state
  s : [0..7] init 0 ;
  // Value of the die (0 = undefined)
  d : [0..6] init 0 ;

  // Flipping the first coin
  [ ] s=0 → 0.5 : (s'=1) + 0.5 : (s'=2) ;
  [ ] s=1 → 0.5 : (s'=3) + 0.5 : (s'=4) ;
  [ ] s=2 → 0.5 : (s'=5) + 0.5 : (s'=6) ;
  [ ] s=3 → 0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1) ;
  [ ] s=4 → 0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3) ;
  [ ] s=5 → 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5) ;
  [ ] s=6 → 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6) ;

  // Absorbing state, after we've decided on the value of the die
  [ ] s=7 → 1.0 : (s'=7) ;

endmodule
```

Let's start by building the model. Select `Model > Build model`, or press F3. We find that the model has 13 states and 20 transitions, which agrees with the DTMC that we've seen (for reference, it's shown on page 750 of [BK08]). Let's now compute the steady state distribution of the model. Select `Model > Compute > Steady-state probabilities`, or press F4. You should be taken to the *Log* tab, and see the following:

```
Probabilities:
7:(7,1)=0.16666650772094727
8:(7,2)=0.16666650772094727
9:(7,3)=0.16666650772094727
10:(7,4)=0.16666650772094727
11:(7,5)=0.16666650772094727
12:(7,6)=0.16666650772094727
```

There will also be a lot of other information, but this is the important output to look at. The six states listed correspond to the states where $s = 7$, and $1 \leq d \leq 6$, and we can see that the probability of being in each state is $\frac{1}{6}$, as we would expect. The probabilities aren't *exactly* $\frac{1}{6}$ because PRISM uses numerical methods to compute them, and so can only calculate each probability up to a certain precision.

Let's now take a look at some transient probabilities. We might want to ask, for example, what the probability distribution is after three time steps. Select `Model > Compute > Transient probabilities`, and enter '3' when prompted for a time. This time, you will see the following output:

```
Probabilities:
1:(1,0)=0.125
2:(2,0)=0.125
7:(7,1)=0.125
8:(7,2)=0.125
9:(7,3)=0.125
10:(7,4)=0.125
11:(7,5)=0.125
12:(7,6)=0.125
```

From this information, we can see that after three time steps, we are either in one of the absorbing states (with an equal probability for each value of the die), or we are either in the state where $s = 1$ or $s = 2$. You should investigate the transient probabilities at different time points, to check that there is always the same probability of choosing each outcome for the die.

Let's now move to the *Properties* tab, and look at model checking some PCTL properties. You can find these in the file `dice.pctl` in the *Mandatory Assignment Materials* directory on CampusNet. We'll start with a simple qualitative property, by asking whether we can be sure that we will eventually get an answer. In other words, we want to know if we eventually reach a state where $s = 7$ with probability one:

```
P>=1[F s=7]
```

If we verify this property, we indeed find that it is true, for all states in the model. Another property we might like to verify is that the dice is *fair*. If this is the case, we should be able to guarantee that, starting in the initial state, the probability of generating an odd number is no greater than one half. We can express this in PRISM as follows:

```
!P>0.5[F (d=1 | d=3 | d=5)]
```

Again, we can verify this, and we see that the property is true.

As discussed in the previous section, it is often more useful in practice to model check *quantitative* properties, which ask what the actual probability of selecting a path that satisfies a given path formula is. For example, this lets us ask what the likelihood of rolling a six is:

```
P=?[F d=6]
```

When we verify the property, PRISM displays a probability, rather than a truth value — in this case, it is 0.16666650772094727, which is not precisely $\frac{1}{6}$ because of rounding errors in the model checker. Remember that the value of a quantitative property in PRISM is by default given for the initial state of the model. If we wanted to calculate this for a different state, such as when $s = 2$, we would write:

```
P=?[F d=6 {s=2}]
```

This time, the answer is 0.33333325386047363 — you should convince yourself that this is correct, by looking back at the DTMC from the lecture.

Experiments in PRISM

One useful feature of PRISM that we haven't yet described is the ability to perform what are called *experiments*. The idea is that we sometimes have a quantitative property that is parameterised, and we would like to know the answer for different values of the parameter. As an example, let's consider the following quantitative PCTL property for our dice model:

```
P=?[F<=t s=7]
```

This says, what is the probability that, starting from the initial state, we will reach a state where $s = 7$ within t time units? If we have a specific value of t, we can use PRISM to model check this property directly, but let's imagine instead that we'd like to plot a graph of the probability of the property holding, for $0 \leq t \leq 10$. We can do this in PRISM quite easily, using the following steps:

1. In the *Constants* panel, directly underneath the *Properties* panel in the *Properties* tab, right-click, and select `Add constant`. Give it a name '`t`', a type '`int`', and leave the *Value* field empty.

2. In the *Properties* panel, enter the above PCTL formula, using the constant '`t`' in place of an actual time bound. In general, you can use constants in any part of the formula, such as for testing the value of a variable.

3. Right-click the property, and select `New experiment`.

4. In the dialog box, choose to enter a range — in this case, with a start value of 0, an end value of 10, and a step size of 1. Make sure that `Create Graph` is selected. This is illustrated in Figure 2.

5. Click `Okay`, and then `Okay` again, when asked about a new graph series. Your screen should now look like Figure 3.

From the graph you produce, you can see how the probability of entering an absorbing state increases at each time step. If you right-click the graph, you can choose to export it as an image.

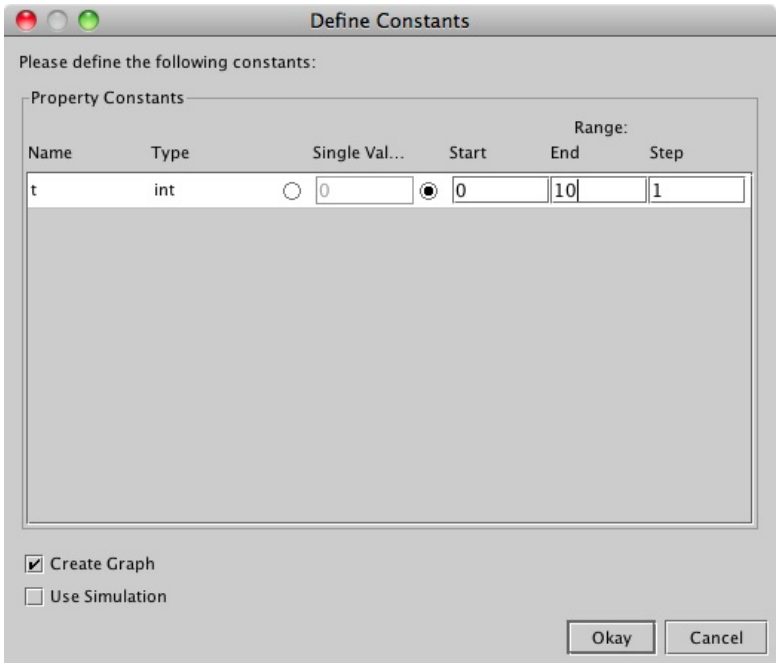

Figure 2: Creating an experiment in PRISM

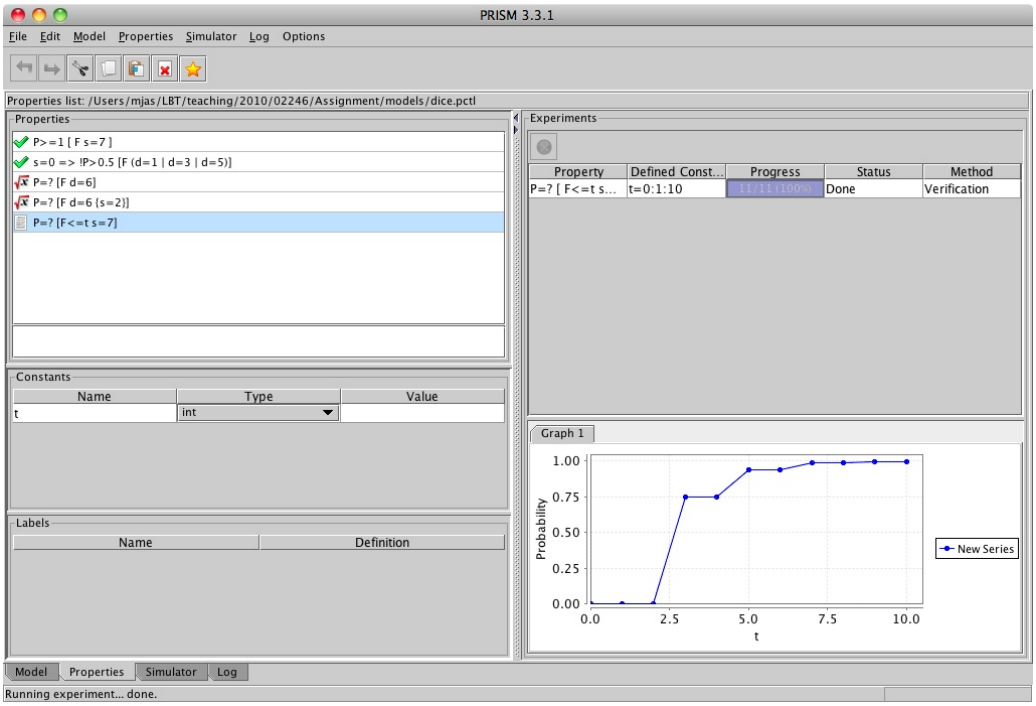

Figure 3: The result of an experiment in PRISM