Probabilistic Automata - continued

We shall be looking at a couple more examples of probabilistic models based on discrete-time Markov chains; and we be looking more closely at how probability of a sequence of actions is computed.

Acknowledgement

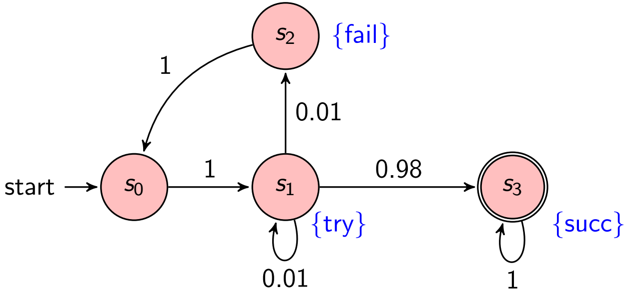
These lectures are based on those of Dave Parker, published on the PRISM web site: see “further reading” at the end of these notes.

Example

The probabilistic automata we have met up until now are called discrete-time Markov chains because their steps are often interpreted as passage of time in discrete steps.

An example of this is the following simple communication protocol:

- Time is *discrete* - it proceeds in 'ticks'
- After one tick, start to send message
- With probability 0.01, the channel is unready; retry after one tick
- With probability 0.98, message sent successfully; stop
- With probability 0.01, send fails; start from beginning



Example (ctd)

- $S = \{s_0, s_1, s_2, s_3\}$; $s_{ini} = s_0$
- $$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  The row and column numbers of P correspond to s_0, s_1, s_2, s_3 respectively. The zero entries are *impossible* transitions: there is no corresponding arrow on the diagram.
- So the matrix is a concise notation for $P(s_0, s_0) = 0, P(s_0, s_1) = 1$, $P(s_1, s_1) = P(s_1, s_2) = 0.1, P(s_1, s_3) = 0.98$, etc.
- Nonzero values of $P()$ label arrows on the graph.
- The atomic propositions are $\{try, succ, fail\}$. $L(s_0) = \emptyset$, $L(s_1) = \{try\}$, $L(s_2) = \{fail\}$, $L(s_3) = \{succ\}$.

Example (ctd)

Exercise:

- Review the die-roll simulation as a discrete-time Markov chain. Could the states themselves may serve as atomic propositions (s means "at s")?
- Can you invent some suitable atomic propositions? - *undecided*, *done*, *threw*1, *threw*2, etc.
- For each $k, L(u_k) = \{u_k, \mathit{undecided}\}$; $L(t_4) = \{t_4, \mathit{done}, \mathit{threw}4\}$ and so forth.
- How would you express 'threw an even'? 'threw and odd or a six'?

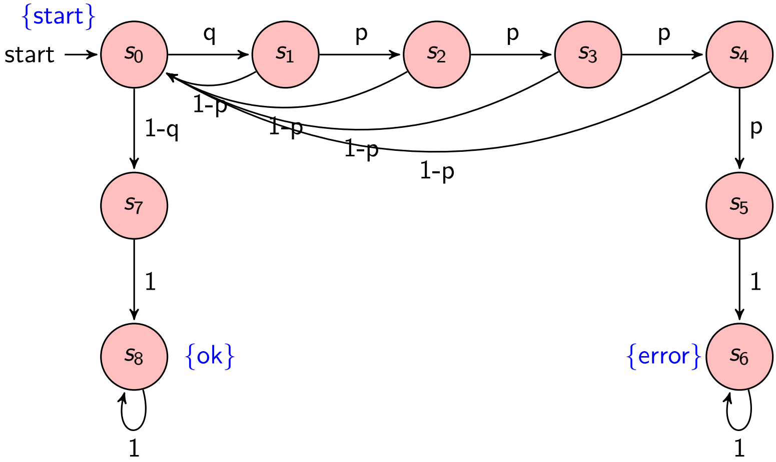
Another example - Zeroconf protocol

"Zero configuration networking" - this is a self-configuration for local ad-hoc networks; it automatically configures a unique IP for new device. The idea:

- 65024 available IP addresses
- A new node picks address U at random and broadcasts "who is using U?"
- A user using U replies; in this case the protocol restarts.
- Message may fail to be sent ...
- so nodes try send multiple (n) probes, waiting after each.

Below is an example with $n = 4$ probes; there are m already-existing nodes on network; p = probability of message loss; $q = m/65024$ = prob that a random new address is already in use.

Zeroconf protocol, ctd



Exercise: write down P for thisMarkov chain. (It is 9x9 and most of the entries are 0)

Probabilities and Paths 1

In the zeroconf example we would like to know what the probability is of ending in $s_8\{ok\}$ having started at s_0,and even if this is 1, a certainty, we would like to know the average or *expected* number of steps it will take. As with a die-throw simulation, there are infinitely many paths from s_0 to s_8.

- By path, we mean an infinite sequence of states $\omega = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots$ such that $\forall k > 0. P(q_{k-1}, q_k) > 0$: all the transition probabilities are positive; there are no 'impossible' steps in the path.
- A *finite* path is a similarly a finite tuple of states $\omega = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n$ with no impossible steps: $P(q_{k-1}, q_k) > 0$ for $k = 1, \ldots, n$.
- Finite path ω_1 is a *prefix* of path ω_2 when $\omega_2 = \omega_1 +$ (concatenated with) an infinite tail.
- For a finite path ω the *cylinder set* $Cyl(\omega) \triangleq$ the set of all inifinite paths which have ω as a prefix.

From now a *path* is inifinite unless specifically described as finite.

Probabilities and Paths 2

We saw in the die-throw simulation that we can compute a probability for a *finite* path $\omega = q_0 \to q_1 \to \ldots \to q_n$ by multiplying the probabilities labelling the transition arrows:
- using the probability matrix P of the Markov chain, this is

- $P_{q_0}(\omega) \triangleq P(q_0, q_1) \times P(q_1, q_2) \times \ldots \times P(q_{n-1}, q_n)$.
    - If $n = 0$ then $P_{q_0}(\omega) = P_{q_0}(q_0) \triangleq 1$.

This is the probability of a particular finite sequence of transitions occurring, given a start in state q_0.

Probabilities and Paths 3

We can now define the probabilities of quite a large class of *sets of paths*.

- For finite path ω starting from q,
  - The cylinder set $Cyl(\omega)$ is the set of all paths with prefix ω;
  - $Pr_q(Cyl(\omega)) \triangleq P_q(\omega)$.
  - Think! this is a consistent definition, but why?
  - How does it make intuitive sense?
- If a set Π of paths from q can be written as a union of *disjoint* (non-overlapping) cylinder sets, then $Pr_q(\Pi)$ is the total of the probabilities of these cylinder sets. Formally
  - if $\Pi = Cyl(\omega_1) \cup Cyl(\omega_2) \cup Cyl(\omega_3) \cup \ldots$ where $Cyl(\omega_i \cap \omega_j) = \emptyset$ for $i \neq j$,
  - then $Pr_q(\Pi) \triangleq Pr_q(\omega_1) + Pr_q(\omega_2) + Pr_q(\omega_3) + \ldots$.
- This is a consistent definition but the sum might be infinite.

Probabilities and Paths 4

A key property of probabilistic systems is *probabilistic reachability*:

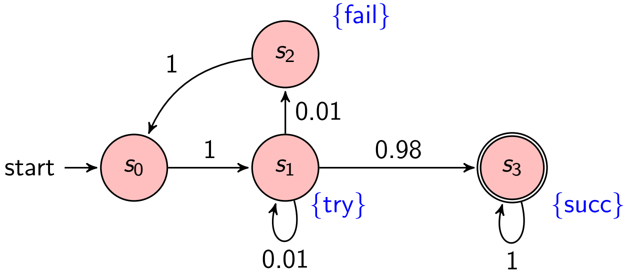
- Given a start state q_0 and a subset of states $T \subseteq S$, what is the probability of a system starting in state q_0 evolving to a state in T?
- The dual is *probabilistic invariance*: the probability of remaining in a set of states $T = 1$ - (the probability of reaching $S - T$).

Probabilistic reachability is computed from probabilities of cylinder sets of paths:

- A path which starts in a state q_0 and visits a state in T belongs to a cylinder set of the form $Cyl(q_0, q_1, ..., q_n)$ where *either* $n = 0, q_0 \in T$, *or else* $n > 0, q_0, ...q_{n-1} \notin T, q_n \in T$.
- These cylinder sets of paths, for various values of n and various vectors of states $q_0, q_1, ..., q_n$ (from q_0), are disjoint and their union, $reach(q_0, T)$, comprises all paths from q_0 which visit T.
- So the probability of reaching T from q_0 = the sum of the probabilities of these cylinder sets, as formulated on the previous slide.

Example: the simple communication protocol again

Refer to slides 2, 3.



Probability of sending failing in first try?

- all such paths $\in Cyl(s_0 s_1 s_2)$.
- $Pr_{s_0}(Cyl(s_0 s_1 s_2)) = P(s_0, s_1)P(s_1, s_2) = 1 \times 0.01 = 0.01$

Paths which are eventually successful with no failures are

- $Cyl(s_0 s_1 s_3) \cup Cyl(s_0 s_1 s_1 s_3) \cup Cyl(s_0 s_1 s_1 s_1 s_3) \cup \ldots.$
- probability $= Pr_{s_0}(s_0 s_1 s_3) + Pr_{s_0}(s_0 s_1 s_1 s_3) + Pr_{s_0}(s_0 s_1 s_1 s_1 s_3) + \ldots$
  $= \sum_{k=0}^{\infty} 1 \times 0.01^k \times 0.98 = 0.98989898\ldots = \frac{98}{99}$

Probabilities and Paths 4

$Pr_{q_0}(reach(q_0, T))$, the probability of a run from state q_0 reaching T, equals

$$\sum_{(q_0 \ldots q_n)} P_{q_0}(q_0, \ldots q_n) = \sum_{(q_0 \ldots q_n)} P(q_0, q_1) \times P(q_1, q_2) \times \ldots \times P(q_{n-1}, q_n)$$

- The sum is taken over all finite paths $\omega = (q_0 \ldots q_n)$ of various lengths n, from q_0 to a state in T ($q_0, \ldots, q_{n-1} \notin T$ and $q_n \in T$).
- $P(q_0, q_1)$ etc are probability matrix entries, remember, and only paths in which they are all > 0 are included.
- If $q_0 \in T$, there is just a path of length $n = 0$ included in the sum: in this case $P_{q_0}(q_0) = 1$ and $Pr_{q_0}(reach(q_0, T)) = 1$ trivially.

This is exactly the calculation we did in the die-throw simulation to obtain the probability $\frac{1}{6}$ for obtaining a 2: ie, for reaching state t_2.

Calculating Reachability Probabilities

To calculate reachability probabilities in practice, we use software.

- The most efficient approach to to compute a *vector* $ProbReach(T)$ of probabilities, containing an entry for each state of the automaton S.
- $ProbReach(T) = x = \langle x_s \rangle_{s \in S}$ where $x_s = Pr_s(reach(s, T))$.
  - If $s \in T$, then $x_s = 1$
  - If $s \notin T$ then $x_s = \sum_{s' \in S} P(s, s')x_{s'}$
  - If T is unreachable from s, then $x_s = 0$

In fact this vector x is the *least fixed point* of a certain function on vectors of probabilities; namely $F : [0, 1]^S \to [0, 1]^S$ defined by $F(y) = z$ where, for $s \in S$,

- if $s \in T$, then $z_s = 1$,
- otherwise $z_s = \sum_{s' \in S} P(s, s')y_{s'}$.

Calculating Reachability Probabilities ctd

This fixed point x can be reached from below:

- Let $x^{(0)} = 0$; ie, for $s \in S, x_s^{(0)} = 0$;
- For $k = 1, 2, 3, \ldots$ let $x^{(k+1)} = F(x^{(k)})$.

This gives an iterative procedure for computing approximations (from below) to $x = \mathit{ProbReach}(T)$:

$$0 = x^{(0)} \leq x^{(1)} \leq x^{(2)} \leq x^{(3)} \leq \ldots \leq x$$

We let the procedure iterate until $x^{(k)}, x^{(k+1)}$ agree to within some pre-defined *tolerance* ϵ.

Calculating Reachability Probabilities - a Java app

An implementation (in Java) accompanies these notes and you can experiment with it.

- Download and unpack `ProbReach.zip`
- Start a terminal session with `ProbReach/` as the working directory.
- Run `java ProbReachUI <file>`
  - This is a graphical interface
  - You need to supply the probabilities matrix in the file:
  - See, for instance, `dieThrowSim.txt`
- $\epsilon = 10^{-16}$ by default in this implementation – the limit of double precision arithmetic.
- The application computes the vector $ProbReach(T)$ of probabilities (slide 13) for $T \subseteq S$.
  - S is the set all states in the model. The application supports up to 64 states - imagine them as $\{s_0, s_1, ...s_{63}\}$. T is represented in the application as a long (64-bit) integer with, for $n = 0...63$, bit $n = 1$ when $s_n \in T$ and bit $n = 0$ when $s_n \notin T$.

Java app - the functon F determined by state set T
(slide 13)

```
/* The function F determined by state-set T, of which we wish to find
 *   the least fixed point.
 * F maps vectors of probabilies (indexed by the states) to same. */
public double[] F(double[] y, long T) {
  if (y.length != nSts) {
    System.err.printf("Probablity vector has wrong size: %d\n", y.length);
    return y;
  }
  double[] ny = new double[nSts];
  double pp;
  for (int i=0; i<nSts; i++) {
    if (isIn(i, T)) {
      ny[i] = 1;
    } else { //ny[i] = sum_j P[i][j].y[j]
      pp = 0.0;
      for (int j=0; j<nSts; j++) {
        pp += P[i][j] * y[j];
      }
      ny[i] = pp;
    }
  }
  return ny;
} //end F(y,T)
```

Java app - computing the least fixed point of F

```
// Compute the least fixed point of F determined by T as above
// For each state s, the probability of reaching T from s = the s-cpt
//   of the vector returned by this computation.
public double[] leastFP(long T) {
  double[] x = new double[nSts],
          nx = new double[nSts];
  for (int i=0; i<nSts; i++)
    x[i] = 0.0;

  int itnNo = 0;
  nx = F(x,T);
  while (diff(nx,x)) {
    itnNo++;
    System.out.printf("%d iterations\r", itnNo);
    x = nx;
    nx = F(x,T);
  }
  System.out.println();
  return x;
} //end leastFP(T)
```

Using the Java app

Try it with the die-throw simulation:

```
$ java ProbReachUI dieThrowSim.txt
```

- $T = \{t_1, t_2, ..., t_6\}$: these are actually states 7,8,9,10,11,12 so as a long integer, $T = 1111110000000_{bin} = 1F80_{hex}$.
- To compute $ProbReach(T)$ takes 56 iterations at this tolerance, and yields vector of 1s as you would expect.
- Try it with various subsets of T. You will get the expected probbilities in the u_0 component: how do you interpret the other component probabilities?
- The application will also calculate the probablities of reaching T from each of the states in $\leq n$ steps – put a value of n in the 'bound' box. Explain the result you get from, say, $n = 9$.

Simple Comms and Zeroconf again

1. Make a probabiities file for the simple commnications protocol of slides 2,3 and investigate it with the `ProbReachUI` application.
2. The Zeroconf protocol (slides 5, 6) has a variable number of states depending on the number of probe messages employed, and other variable parameters - the number of pre-existing nodes and the probability of measage loss.
   Java class `GenZeroconf` is an application thay generates the (probability matrix of the) Zeroconf protocol for a given set of values of the parameters.
   Use this to experiment with reachability probability vectors for the Zeroconf protocol.

PRISM again

- See if you can use PRISM to obtain the reachbility results you have obtained with `ProbReachUI`.
- You will need to check out *probabilistic computation tree logic* (PCTL). More on this next week!

Further Reading

- The material in this and a previous lecture are covered by Dave Parker's lecture 2: see http://www.prismmodelchecker.org/lectures/pmc/).
- A good theory reference is chapter 10 of Principles of Model Checking by C Baier and J-P Katoen (MIT Press, 2008)