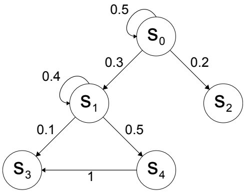
Verification and Probabilistic Logic Programming

C. R. Ramakrishnan, Andrey Gorlin

Stony Brook University

ICLP 2016 Autumn School

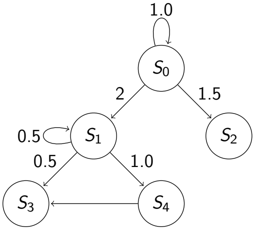
System Description: I



Discrete-Time Markov Chains (DTMCs)

- Probabilistic automata where transitions out of a state are governed by a discrete *distribution*.
- Sets of atomic propositions may be associated with individual states.
- Next-state distribution depends only on the current state and not on the past (given the current state).
- A variety of process languages are "compiled" into DTMCs.

System Description: II

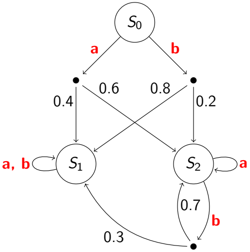


Continuous-Time Markov Chains (CTMCs)

- Probabilistic automata where transitions out of a state have associated *rates*.
- The rates govern the time at which a transition "fires" (distributed exponentially).
- An execution is extended by taking first transition to fire from current state.

- Analysis of CTMCs is often done by analyzing an associated DTMC.
- We will focus on discrete-time probabilistic systems in this lecture.

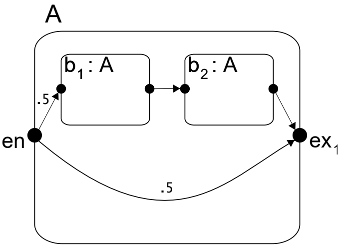
System Description: III



Markov Decision Processes (MDPs)

- Discrete-time probabilistic automata where each state has a set of *uniquely labeled* transitions.
- Each transition specifies a *distribution* of destination states (in general, not just a single state).
- Combines non-deterministic choice of transitions with probabilistic choice of destination based on a chosen transition.
- Used widely to model behaviors of agents.

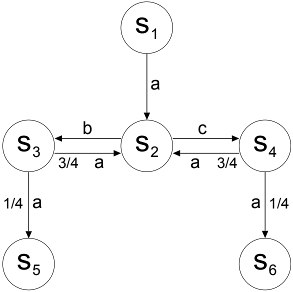
System Description: IV



Recursive Markov Chains (RMCs)

- Models of Probabilistic Programs: Extension of DTMCs with “Calls” to model non-tail-recursive procedures.
- Each RMC has a distinguished “Entry” state (which is reached when that RMC is “called”).
- Each RMC may have one or more “Exits”, which can be used to model return values.

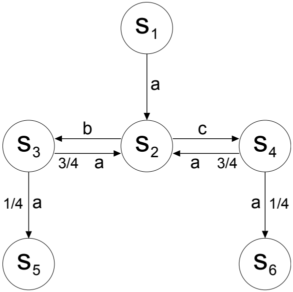
Reactive Probabilistic Labeled Transition Systems (RPLTS)



- Automata has finite number of states.
- Each state offers a finite number of labeled *actions*.
- Each action has a *distribution* of states: taking an action chooses a destination state according to the given distribution.
- Actions are triggered by an external agent; the system *reacts* to actions.

[Cleaveland, Iyer & Narasimha, TCS'05]

RPLTS vs MDPs



- RPLTS and MDPs are structurally identical but are interpreted differently.
- RPLTS semantics is given in terms of a distribution over computation trees, where
  - Probabilistic choices are first resolved in order to construct computation trees, and
  - The trees, in turn, capture the available non-deterministic choices.

Properties of Probabilistic Systems

- **Reachability:** Find the probability that a run of the system eventually reaches a given state.
  Related problem: **termination**.
- **Probabilistic Temporal Logics:** Formulae in such logics express complex temporal conditions on the behaviors of a system. Behaviors may be
  - *runs*: linear-time logics
  - *trees*: branching-time logics

  The problem is to find the probability of behaviors that satisfy the temporal conditions.
- **Optimization:** For MDPs and other systems with non-determinism, find the *min.* or *max.* probability of a specified behavior.

Probabilistic Computation Tree Logic (PCTL)

- PCTL is a logic for specifying properties of DTMCs.

$$\begin{array}{lll} \varphi & \rightarrow & p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \\ & \mid & Pr(\phi) > b \mid Pr(\phi) \geq b \end{array}$$

Propositions, logical connectives; State formulae

$$\phi \rightarrow X\varphi \mid \varphi_1 \; U \; \varphi_2$$

Path formulae

- State formulas are non-probabilistic; path formulas have associated probabilities.
- Used as the property specification language by many systems, including the Prism Model Checker.
- Example: $Pr(p \; U \; q) > 0.75$
  Is the probability of a run where p holds until q more than 0.75?

Generalized Probabilistic Logic (GPL)

- An expressive, mu-calculus-based, logic for branching-time probabilistic processes.
- Semantics of GPL is given in terms of computation trees of RPLTSs.
- This logic is strictly more expressive than PCTL*.
- Reachability and termination in RMCs can be reduced to GPL model checking over RPLTSs.
- We can construct a model checker for GPL by directly encoding its semantics as a probabilistic logic program.

GPL

- Usual mu-calculus-like modalities and fixed points (called "state formulae") in GPL.
- State formulae, ϕ, have a boolean interpretation:

$$\phi = \phi \vee \phi \;|\cdots|\; \mathtt{pr}^{>B}\psi \,|\, \mathtt{pr}^{\geq B}\psi \,|\; \cdots \text{ propositions}\ldots$$

- Fuzzy formulae ψ, analogous to PCTL path formulae, have probabilistic interpretation:

$$\psi = \psi \vee \psi \;|\; \psi \wedge \psi \;|\; \langle a \rangle \psi \;|\; [a]\psi \;|\; \phi \;|\; X$$

- *Alternation-free* fixed point equations of the form $X =_\mu \psi$ and $X =_\nu \psi$.

System Definitions as PLP

- Recall encoding DTMCs in PRISM:

  % DTMC Transition Relation

  ```
  trans(S, I, T) :- msw(t(S), I, T).
  ```

  where switch `t(`s`)` encodes the transition distribution from state s.
- For MDPs and RPLTSs, each action gives a distribution. This is encoded as facts of the following form:

  % MDP/RPLTS Action Definitions

  `action(`S, A, SW`)`

  where "S" is the source state, "A" is a transition label, and SW is a **switch** whose distribution models the action's distribution.
- MDP/RPLTS transitions are defined by:

  % DTMC Transition Relation

  ```
  trans(S, A, I, T) :- action(S, A, SW), msw(SW, I, T).
  ```

Encoding the PCTL Model Checker

State Formulae

```
% Propositions
models(S, prop(P)) :- holds(S, P).

% Logical connectives
models(S, neg(F)) :- tnot models(S, F).
models(S, and(F1, F2)) :- models(S, F1), models(S, F2).

% Path Quantifiers
models(S, pr(F, gt, B)) :-
    prob(pmodels(S, F), P),
    P > B.
models(S, pr(F, geq, B)) :-
    prob(pmodels(S, F), P),
    P >= B.
```

Encoding the PCTL Model Checker

Path Formulae

- Note that *X* and *U* operators will access the transition relation.
- Outcomes of a transition at different time steps need to be distinguished.

```
% Add extra temporal argument
pmodels(S, F) :- pmodels(S, F, _).

% Next
pmodels(S, next(F), H) :- trans(S, H, T), models(T, F).
```

Encoding the PCTL Model Checker

Path Formulae (contd.)

```
% Until (base case):
pmodels(S, until(F1, F2), H) :-
     pmodels(S, or(F2, and(F1, next(until(F1,F2))))).

% Until (unrolled, recursive case)
pmodels(S, until(F1, F2), H) :-
    models(S, F1), trans(S, H, T),
    pmodels(T, until(F1, F2), next(H)).

% Note the temporal argument in pmodels/3:
temporal(pmodels/3-3).
```

Model Checking in PLP

- Semantics of the (probabilistic) temporal logic is encoded directly as a Probabilistic Logic Program.
- Note that probabilistic temporal logics use standard temporal constructs to specify the behavior to be observed;
  - And simply query the probability of specified behaviors
- Hence it is not surprising that the encoding of a probabilistic model checker is very similar to the non-probabilistic case.

GPL Model Checker

Fuzzy (Path) Formulae

```
% State formulae
pmodels(S, sf(SF), H) :-
   smodels(S, SF).

% Logical Connectives
pmodels(S, and(F1,F2), H) :-
   pmodels(S, F1, H),
   pmodels(S, F2, H).
pmodels(S, or(F1,F2), H) :-
   pmodels(S, F1, H);
   pmodels(S, F2, H).

% Diamond Modality
pmodels(S, diam(A, F), H) :-
    action(S, A, SW),
    msw(SW, H, T),
    pmodels(T, F, [T,SW|H]).
```

- RPLTSs semantics is a distribution of computation trees.
- Each distinct history of actions taken determines a root-to-leaf path in a tree.
- Each distinct history results in a distinct instance of random variables (for choosing the next destination).
- This is reflected in the treatment of instance variables in the “diamond” clause.

GPL Model Checker

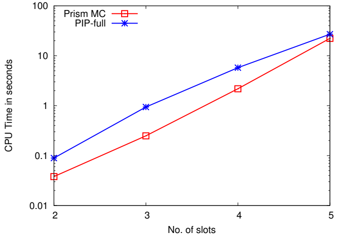
Boxes and Fixed Points

```
% Box formulae
pmodels(S, box(A, F), H) :-
    findall(SW, action(S,A,SW), L),
    all_pmodels(L, S, F, H).

% Least fixed point formulae
pmodels(S, form(X), H) :-
    lfp(X, F), pmodels(S, F, H).

% Greatest fixed point formulae
pmodels(S, form(X), H) :-
    gfp(X, F), negate(F, NF),
    tnot pmodels(S, NF, H).

all_pmodels([], _, _, _H).
all_pmodels([SW|Rest], S, F, H) :-
    msw(SW, H, T),
    pmodels(T,F,[T,SW|H]),
    all_pmodels(Rest, S, F, H).
```

- "Box" modality universally quantifies over all possible actions with a given label.
- LFP and GFP formulae treated the same was as for the non-probabilistic case.
- Model checker for *state formulae* are straightforward and omitted.

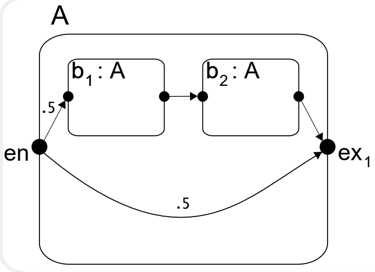
Performance Impact of PLP

6 processes:



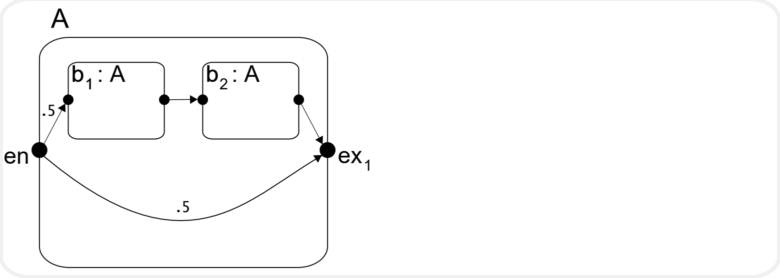
- Time performance is compared with that of the Prism Model Checker.
- System specified using Prism's modeling language (Reactive Modues, RM).
- Example shown:
  - *System*: Synchronous Leader Election protocol
  - *Property*: "eventually a leader is elected" (reachability).
- Model checking times are within a factor of 3 (note log scale).

Reachability in RMCs: I



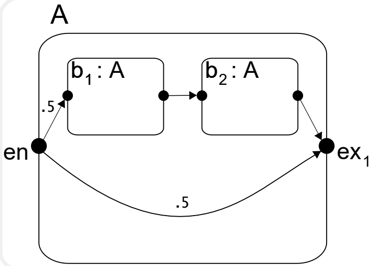
- "Call" to A enters at *en*.
- With 0.5 probability, we immediately return (leave at ex_1)
- With 0.5 probability, we call A recursively, twice.
- What is the probability that some call to A will reach ex_1?

Reachability in RMCs: II



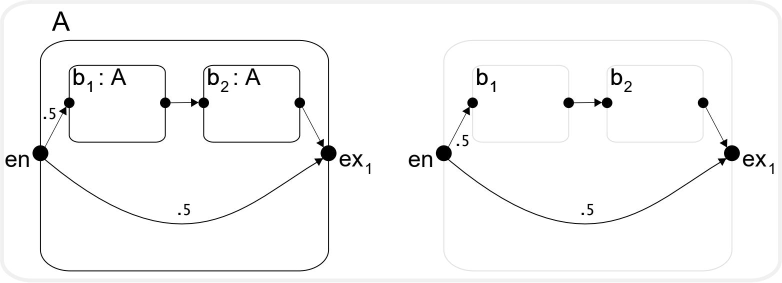
- Specialized techniques have been developed to answer reachability and termination questions.
- These techniques generate and solve systems of monotone polynomial (possibly non-linear) equations.

Reachability in RMCs as GPL Model Checking



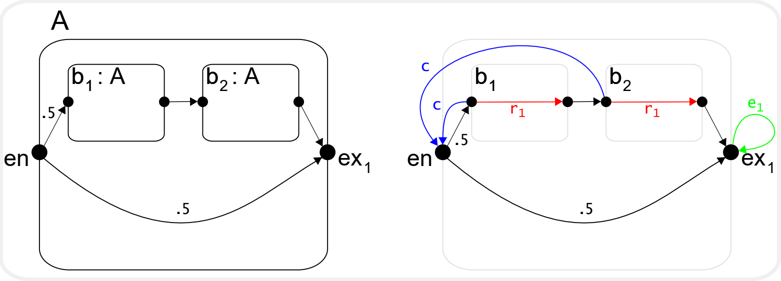
- Construct an RPLTS from RMC by replacing calls with *c*, *e*, and *r* transitions.

Reachability in RMCs as GPL Model Checking



- Construct an RPLTS from RMC by replacing calls with c, e, and r transitions.

Reachability in RMCs as GPL Model Checking



- Construct an RPLTS from RMC by replacing calls with c, e, and r transitions.
- Construct a GPL formula to match calls to returns:
- X_1: eventually exit ex_1 is reached:

$$X_1 =_{\mu} \langle e_1\rangle \mathtt{tt} \vee \langle p\rangle X_1 \vee (\langle c\rangle X_1 \wedge \langle r_1\rangle X_1)$$

RMCs and GPL

- Given an RMC, we uniquely number each exit state.
- Consider an RMC with n exits.
- The property "Exit ex_i is eventually reached when a recursive procedure is entered" is given by GPL formula:

$$
\begin{array}{lll}
X_i & =_\mu & \langle e_i \rangle \mathtt{tt} \quad \vee \quad \langle p \rangle X_i \\
 & & \vee\ (\langle c \rangle X_1 \ \wedge\ \langle r_1 \rangle X_i) \\
 & & \vee\ (\langle c \rangle X_2 \ \wedge\ \langle r_2 \rangle X_i) \\
 & & \vdots \\
 & & \vee\ (\langle c \rangle X_n \ \wedge\ \langle r_n \rangle X_i)
\end{array}
$$

Markov Decision Processes (MDPs)

- MDP looks very similar to an RPLTS: actions on states that have a distribution of destination states.
- Semantics is different in two ways:
  - States have "*rewards*", and induce rewards on paths.
  - Schedulers dictate actions taken at each state.
- Interesting problem: find an *optimal* scheduler that maximizes the expected reward.

Committed Choice

- A scheduler commits an MDP to take a specific action at some point in its run.
- Analogous to `msw`, we introduce `nd(X, I, V)` to choose from a set and commit to that choice.
  - `X` is a discrete-valued choice process
  - `V` is a value generated by the choice process
  - `I` is the *instance number*.
- Example: `nd(`s_2`, 0, X)` with `values(`s_2`, [b,c])` will `X` to `b` in one set of worlds, and to `c` in another.
- Distribution semantics is naturally extended: the meaning of a program is a distribution of **sets of** models.

Committed Choice (contd.)

```
q(Y) :- nd(f, 0, X),
        msw(X, 0, Y).
values(f, [a,b]).
values(a, [t,f]).
values(b, [t,f]).
set_sw(a, [0.3, 0.7])
set_sw(b, [0.6, 0.4])
```

```
?-  prob(q(t), P).

P = 0.3
;
P = 0.6
```

- Probability of an answer is computed separately for each distinct set of committed choices.
- For recursive programs (MDPs), each set of committed choices will yield a set of linear equations, whose least solution will be the corresponding probability.
- Expected rewards can be computed analogously.
- We can find optimal probabilities (and, similarly, optimal expected reward) by pushing a `max` operation into the equations themselves.

Model Checking as Query Evaluation

Mobile Ad-Hoc Networks

Parameterized Systems

Multi-Agent Systems

Model Checkers

`Infinite-State Systems`

π-Calculus

- Model checkers were built from high-level specifications of the semantics of non-probabilistic temporal logics
  - Used the termination and sharing properties of *tabling*-based query evaluation.

Model Checking as Query Evaluation

Mobile Ad-Hoc Networks

Parameterized Systems

Multi-Agent Systems

Model Checkers

`Infinite-State Systems`

π-Calculus

Probabilistic Systems

- Model checkers were built from high-level specifications of the semantics of non-probabilistic temporal logics
  - Used the termination and sharing properties of *tabling*-based query evaluation.
- Model checkers for probabilistic systems build on these results.
  - Used a temporal probabilistic inference algorithm