Probabilistic Tabled Logic Programming with Application to Model Checking

C. R. Ramakrishnan

Stony Brook University

ICLP 2013
Executable Specification of Operational Semantics

\[
\begin{align*}
    e_1 & \rightarrow e_1' \\
    (e_1 e_2) & \rightarrow (e_1' e_2)
\end{align*}
\]

\[
\begin{align*}
    e_2 & \rightarrow e_2' \\
    (v_1 e_2) & \rightarrow (v_1 e_2')
\end{align*}
\]

\[
\begin{align*}
    (\lambda x. e_1) v_2 & \rightarrow [x \mapsto v_2] e_1
\end{align*}
\]

\[
\begin{align*}
    \text{step(app}(E1, E2), \text{app}(E1P, E2)) & : - \\
    \text{step}(E1, E1P).
\end{align*}
\]

\[
\begin{align*}
    \text{step(app}(V1, E2), \text{app}(V1, E2P)) & : - \\
    \text{isValue}(V1), \\
    \text{step}(E2, E2P).
\end{align*}
\]

\[
\begin{align*}
    \text{step(app}(\lambda(X, E1), V2), E2) & : - \\
    \text{isValue}(V2), \\
    \text{subst}(X, V2, E1, E2).
\end{align*}
\]

\[
\begin{align*}
    \text{isValue}(\lambda(_, _)).
\end{align*}
\]

[Call-By-Value Lambda Calculus]
Substitution

\[
\begin{align*}
[x \mapsto s]x &= s \\
[x \mapsto s]y &= y & \text{if } y \neq x \\
[x \mapsto s](\lambda y. t) &= \lambda y. [x \mapsto s]t & \text{if } x \neq y \text{ and } y \not\in \text{fv}(s) \\
[x \mapsto s](t_1 \ t_2) &= ([x \mapsto s]t_1) ([x \mapsto s]t_2)
\end{align*}
\]
Substitution

\[
\begin{align*}
[x \mapsto s] x &= s \\
[x \mapsto s] y &= y & \text{if } y \neq x \\
[x \mapsto s](\lambda y. t) &= \lambda y. [x \mapsto s] t & \text{if } x \neq y \text{ and } y \notin \text{fv}(s) \\
[x \mapsto s](t_1 t_2) &= ([x \mapsto s] t_1)([x \mapsto s] t_2)
\end{align*}
\]

- This definition becomes complete only when we consider \(\alpha\)-renaming.
- We can program \(\alpha\)-renaming explicitly, or better still...
Substitution

\[ [x \mapsto s]x = s \]
\[ [x \mapsto s]y = y \quad \text{if } y \neq x \]
\[ [x \mapsto s](\lambda y. \ t) = \lambda y. [x \mapsto s]t \quad \text{if } x \neq y \text{ and } y \not\in \text{fv}(s) \]
\[ [x \mapsto s](t_1 \ t_2) = ([x \mapsto s]t_1) ([x \mapsto s]t_2) \]

- This definition becomes complete only when we consider \(\alpha\)-renaming.
- We can program \(\alpha\)-renaming explicitly, or better still...
- With suitable restrictions on the way \(\lambda\)-terms are written,
  - represent variables in lambda-terms with logical variables, and
  - use the “standardization” done by resolution to perform the needed \(\alpha\)-renaming.
- We used such a strategy to encode model checkers for the \(pi\)-calculus [Yang et al, VMCAI’03].
Executable Specification of Abstract Semantics

\[ \begin{align*}
& \frac{p = \& q}{p \rightarrow q} \\
& \frac{p = q \quad q \rightarrow r}{p \rightarrow r} \\
& \frac{p = \ast q \quad q \rightarrow r \quad r \rightarrow s}{p \rightarrow s} \\
& \frac{\ast p = q \quad p \rightarrow r \quad q \rightarrow s}{r \rightarrow s}
\end{align*} \]

\[\begin{align*}
\text{pts}(P, Q) & : - \\
& \text{stmt}(v(P), \text{addr}(Q)).
\end{align*}\]

\[\begin{align*}
\text{pts}(P, R) & : - \\
& \text{stmt}(v(P), v(Q)), \\
& \text{pts}(Q, R).
\end{align*}\]

\[\begin{align*}
\text{pts}(P, S) & : - \\
& \text{stmt}(v(P), \text{star}(Q)), \\
& \text{pts}(Q, R), \text{pts}(R, S).
\end{align*}\]

\[\begin{align*}
\text{pts}(R, S) & : - \\
& \text{stmt}(\text{star}(P), v(Q)), \\
& \text{pts}(P, R), \\
& \text{pts}(Q, S).
\end{align*}\]

[Anderson’s Context-Insensitive Points-To Analysis]
Demand-Driven Analysis

Compute only the information necessary to determine the \textit{may-point-to} set of $x$. [Heinze et al., PLDI 2001]

- Tabled query evaluation is naturally demand-driven, but \dots
Demand-Driven Analysis

Compute only the information necessary to determine the *may-point-to* set of \( x \). [Heinze et al., PLDI 2001]

- Tabled query evaluation is naturally demand-driven, but ...
- Clauses of the form \( \text{pts}(R, S) :- \text{stmt}(\text{star}(P), v(Q)), \ldots \) lead to generate-and-test evaluation.

\[ \text{pts}(R, S) :- \text{stmt}(\text{star}(P), v(Q)), \text{pts}(P, R), \text{pts}(Q, S). \]
\[ \Rightarrow \text{pts}(R, S) :- \text{ptb}(R, P), \text{stmt}(\text{star}(P), v(Q)), \text{pts}(Q, S). \]

[PPDP’05]
Demand-Driven Analysis

Compute only the information necessary to determine the *may-point-to* set of \( x \). [Heinze et al., PLDI 2001]

- Tabled query evaluation is naturally demand-driven, but ...
- Clauses of the form \( \text{pts}(R, S) :- \text{stmt}(\text{star}(P), v(Q)), \) ... lead to generate-and-test evaluation.
- **Trick:** replicate *points-to* (pts) as *pointed-to-by* (ptb).

\[
\begin{align*}
\text{pts}(R, S) :- & \quad \text{stmt}(\text{star}(P), v(Q)), \\
& \quad \text{pts}(P, R), \\
& \quad \text{pts}(Q, S).
\end{align*}
\]

\[
\begin{align*}
\text{pts}(R, S) :- & \quad \text{ptb}(R, P), \\
& \quad \text{stmt}(\text{star}(P), v(Q)), \\
& \quad \text{pts}(Q, S).
\end{align*}
\]

[PPDP’05]
**Incremental Evaluation**

- Computing changes to query answers for definite programs when rules/facts are *added* is relatively easy.
  - Semi-naive and tabling are naturally incremental w.r.t. addition of clauses.

- Computing changes when clauses are *deleted* is harder:
  - DRed [Gupta et al, SIGMOD’93], and similar algorithms in model checking [Sokolsky & Smolka, CAV’94] and program analysis [e.g., Yur et al, ICSE’99] have been proposed for this problem.
  - DRed is prohibitively expensive in practice.
Use of **Support Graphs**, to store dependency between query answers and clauses/facts, makes DRed feasible [Saha & R., ICLP’03].

Application to *incremental program analysis* [Saha & R. PPDP’05]

*Symbolic* support graphs significantly reduce memory requirements for certain classes of programs [Saha & R., ICLP’05].

Subsequent generalization to handle updates [ICLP’06], and Prolog [PADL’06]
Executable Specification of Semantic Equations

\[ [\cdot] \] is the **smallest** set such that:

\% \[ [p] \] = states satisfying prop. \( p \).
\[ [p] = \{ s | p \in AP(s) \} \]

\% Conjunction:
\[ [\varphi_1 \land \varphi_2] = [\varphi_1] \cap [\varphi_2] \]

\% \[ [EF \ f] = \]
\% \{ s | \exists t. \ s \xrightarrow{*} t \text{ and } t \in [f] \}\n\[ [EF \varphi] = [\varphi] \]
\[ \cup \{ s | \exists t. \ s \rightarrow t, t \in [EF \varphi] \} \]

\[ \vdots \]

- \[ \text{models}(S,\text{prop}(P)) :- \]
  \[ \text{holds}(S, P). \]
- \[ \text{models}(S,\,\text{and}(F1,F2)) :- \]
  \[ \text{models}(S, F1), \text{models}(S, F2). \]
- \[ \text{models}(S, \,\text{ef}(F)) :- \]
  \[ \text{models}(S, F). \]
- \[ \text{models}(S, \,\text{ef}(F)) :- \]
  \[ \text{trans}(S, T), \text{models}(T, \,\text{ef}(F)). \]
- \[ \text{models}(S, \,\text{af}(F)) :- \]
  \[ \text{models}(S, F). \]
- \[ \text{models}(S, \,\text{af}(F)) :- \]
  \[ \text{findall}(T, \text{trans}(S, T), L), \]
  \[ \text{all_models}(T, \,\text{af}(F)). \]
- \[ \ldots \]

*[Computation Tree Logic’s Semantics (Fragment)]*
Model Checking and Program Analysis as Query Evaluation

- Mobile Ad-Hoc Networks
- Parameterized Systems
- Multi-Agent Systems
- Model Checkers
- Infinite-State Systems
- $\pi$-Calculus
- Incremental Program Analyzers
- Program Analyzers
- Alias Analysis of C Programs
- Bisimulation Checkers
- Other Analyzers
- Security Policy Analyzers

C. R. Ramakrishnan
Probabilistic Tabled Logic Programming
ICLP 2013 9 / 41
Model Checking and Program Analysis as Query Evaluation

- Mobile Ad-Hoc Networks
- Parameterized Systems
- Multi-Agent Systems
- Model Checkers
- Infinite-State Systems
- $\pi$-Calculus
- Probabilistic Systems

- Incremental Program Analyzers
- Program Analyzers
- Alias Analysis of C Programs

- Bisimulation Checkers
- Other Analyzers
- Security Policy Analyzers

C. R. Ramakrishnan

Probabilistic Tabled Logic Programming

ICLP 2013
Logic Programs

Program Rules

\[ + \quad \models \quad \text{Query Answers} \]

Facts
Probabilistic Logic Programs

Program Rules

+ 

Probabilistic Facts

Query Answers

The PRISM language and system [Sato and Kameya '97]
Probabilistic Logic Programs

Program Rules

\[ + \quad \models \quad \text{Query Answers} \]

The PRISM language and system [Sato and Kameya ’97]
A language for probabilistic logic programming with system for inference and parameter learning (Sato et al, since ’99).

- Logic programs with a set of **probabilistic facts**: \( \text{msw}(X, I, V) \), where
  - \( X \) is a discrete-valued random process
  - \( V \) is a value generated by the random process
  - \( I \) is the *instance number*, distinguishing different trials.

- Random variables generated by the same random process are i.i.d.
- Random variables generated by distinct random processes are independent.
- Has a well-defined model-theoretic (*distribution*) semantics, and an operational semantics based on tabled resolution.
% "a" is a boolean random process
p(X) :- msw(a, 0, X),
       msw(a, 1, Y),
       X=Y.
values(a, [t,f]).
set_sw(a, [0.3,0.7])

Outcomes of random processes define worlds. The probability of a world is assigned based on the probabilities of the outcomes in the world. In each world, \texttt{msw}s form a set of logical (non-probabilistic) facts. Distribution over least models: the least model in each world is assigned the probability of that world.
% “a” is a boolean random process

\[
p(X) :- \text{msw}(a, 0, X), \\
     \text{msw}(a, 1, Y), \\
     X = Y.
\]

\[
\text{values}(a, [t, f]). \\
\text{set\_sw}(a, [0.3, 0.7])
\]

---

**Worlds:**

<table>
<thead>
<tr>
<th>msw(a,0,t)</th>
<th>msw(a,0,f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>msw(a,1,t)</td>
<td>msw(a,1,f)</td>
</tr>
</tbody>
</table>

Outcomes of random processes define worlds.
% “a” is a boolean random process

\[ p(X) :- \text{msw}(a, 0, X), \]
\[ \text{msw}(a, 1, Y), \]
\[ X=Y. \]

\text{values}(a, [t,f]).

\text{set}\_\text{sw}(a, [0.3,0.7])

---

\textbf{Worlds:}

\begin{tabular}{ccc}
\text{msw}(a,0,t) & msw(a,0,t) \\
0.09 & 0.21 \\
\text{msw}(a,1,t) & msw(a,1,f) \\
\text{msw}(a,0,f) & msw(a,0,f) \\
0.21 & 0.49 \\
\text{msw}(a,1,t) & msw(a,1,f) \\
\end{tabular}

- Outcomes of random processes define worlds.
- The probability of a world is assigned based on the probabilities of the outcomes in the world.
% "a" is a boolean random process

\[ p(X) :\quad \text{msw}(a, 0, X), \]
\[ \quad \text{msw}(a, 1, Y), \]
\[ \quad X=Y. \]

values(a, [t,f]).

set_sw(a, [0.3,0.7])

<table>
<thead>
<tr>
<th>Worlds:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>msw(a,0,t)</td>
<td>msw(a,0,t)</td>
<td>0.09</td>
</tr>
<tr>
<td>msw(a,1,t)</td>
<td>msw(a,1,f)</td>
<td>0.21</td>
</tr>
<tr>
<td>msw(a,0,f)</td>
<td>msw(a,0,f)</td>
<td></td>
</tr>
<tr>
<td>msw(a,1,t)</td>
<td>msw(a,1,f)</td>
<td>0.49</td>
</tr>
</tbody>
</table>

- Outcomes of random processes define worlds.
- The probability of a world is assigned based on the probabilities of the outcomes in the world.
- In each world, \( \text{msw} \)s form a set of logical (non-probabilistic) facts.
% "a" is a boolean random process
p(X) :- msw(a, 0, X),
       msw(a, 1, Y),
       X=Y.
values(a, [t,f]).
set_sw(a, [0.3,0.7])

Models:

<table>
<thead>
<tr>
<th>msw(a,0,t)</th>
<th>msw(a,0,t)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>msw(a,0,t)</td>
</tr>
<tr>
<td></td>
<td>msw(a,1,t)</td>
</tr>
<tr>
<td></td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>msw(a,1,f)</td>
</tr>
<tr>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>p(t)</td>
<td>msw(a,0,f)</td>
</tr>
<tr>
<td></td>
<td>msw(a,0,f)</td>
</tr>
<tr>
<td></td>
<td>msw(a,1,t)</td>
</tr>
<tr>
<td></td>
<td>msw(a,1,f)</td>
</tr>
<tr>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>0.49</td>
</tr>
<tr>
<td>p(f)</td>
<td></td>
</tr>
</tbody>
</table>

- Outcomes of random processes define worlds.
- The probability of a world is assigned based on the probabilities of the outcomes in the world.
- In each world, msws form a set of logical (non-probabilistic) facts.
- Distribution over least models: the least model in each world is assigned the probability of that world.
Probabilistic Logic Programs: Background

- Logic-based representation of statistical models
  - Examples include BLPs (Kersting and De Raedt, '00), PRMs (Friedman et al, '99), MLNs (Richarson and Domingos, '06).
  - The underlying statistical network, derived from logical/statistical specifications, is finite.
- Statistical inference over proof structures
  - Conservative extension to traditional logic programs, with explicit or implicit use of random variables and processes.
  - Examples include PRISM (Sato and Kameya, '99), ICL (Poole, '93), CLP(BN) (Santos Costa et al, '03), ProbLog (De Raedt et al, '07), LPAD (Vennekens et al, '09).
  - In terms of expressive power, PRISM, ProbLog and LPAD coincide; however, they use different inference procedures.
% Finite Mixture Model

\[ q(Y) :\sim msw(a, 0, X), \]
\[ msw(b(X), 0, Y). \]

values(a, [t,f]).
values(b(_, [t,f]).
set_sw(a, [0.3,0.7])
set_sw(b(t), [0.6,0.4])
set_sw(b(f), [0.5,0.5])
% Finite Mixture Model

\[
\text{q(Y)} :- \ msw(a, 0, X), \\
\hspace{1cm} msw(b(X), 0, Y).
\]

\[
\text{values(a, [t,f]).} \\
\text{values(b(_), [t,f]).} \\
\text{set\_sw(a, [0.3,0.7])} \\
\text{set\_sw(b(t), [0.6,0.4])} \\
\text{set\_sw(b(f), [0.5,0.5])}
\]

Explanations and Probabilities

\[
\begin{array}{c}
\text{q(t)} \\
\hspace{1cm} msw(a, t) \quad msw(a, f) \\
\hspace{2cm} 0.3 \\
\hspace{3.5cm} msw(b(t), t) \quad msw(b(f), t) \\
\hspace{5cm} 0.6 \\
\hspace{7cm} 0.18
\end{array}
\]
% Finite Mixture Model

\[ q(Y) :\neg msw(a, 0, X), \\
msw(b(X), 0, Y). \]

\[
\text{values}(a, [t,f]). \\
\text{values}(\_b(\_), [t,f]). \\
\text{set}_sw(a, [0.3,0.7]) \\
\text{set}_sw(b(t), [0.6,0.4]) \\
\text{set}_sw(b(f), [0.5,0.5])
\]

Explanations and Probabilities

\[ q(t) \]

\[
\begin{array}{c}
\text{msw}(a, t) \\
0.3 \\
\text{msw}(b(t), t) \\
0.6 \\
0.18 \\
\end{array}
\begin{array}{c}
\text{msw}(a, f) \\
0.7 \\
\text{msw}(b(f), t) \\
0.5 \\
0.35 \\
\end{array}
\]
% Finite Mixture Model
q(Y) :- msw(a, 0, X),
      msw(b(X), 0, Y).

values(a, [t,f]).
values(b(_), [t,f]).
set_sw(a, [0.3,0.7])
set_sw(b(t), [0.6,0.4])
set_sw(b(f), [0.5,0.5])

Explanations *and Probabilities*

q(t) 0.53

msw(a, t) 0.3
msw(b(t), t) 0.6

msw(a, f) 0.7
msw(b(f), t) 0.5

0.18 0.35
Evaluation in PRISM — II

- **Explanation** of an answer: At a high level, the set of msw's used in a derivation of the answer.
Evaluation in PRISM — II

- **Explanation** of an answer: At a high level, the set of msw’s used in a derivation of the answer.
- The probability of an explanation is the product of the probabilities of random variables in the explanation.
Evaluation in PRISM — II

- **Explanation** of an answer: At a high level, the set of msw’s used in a derivation of the answer.

- The probability of an explanation is the product of the probabilities of random variables in the explanation.
  - If the msw’s in a derivation are all independent, then the probability of the explanation can be computed without materializing it.
    [Independence assumption]
Evaluation in PRISM — II

• **Explanation** of an answer: At a high level, the set of msw’s used in a derivation of the answer.

• The probability of an explanation is the product of the probabilities of random variables in the explanation.
  
  • If the msw’s in a derivation are all independent, then the probability of the explanation can be computed without materializing it.  
    
    [Independence assumption]

• The probability of an answer is the probability of the set of explanations of the answer.
Evaluation in PRISM — II

- **Explanation** of an answer: At a high level, the set of msw’s used in a derivation of the answer.

- The probability of an explanation is the product of the probabilities of random variables in the explanation.
  - If the msw’s in a derivation are all independent, then the probability of the explanation can be computed without materializing it.
    
    [Independence assumption]

- The probability of an answer is the probability of the set of explanations of the answer.
  - If explanations are pairwise mutually exclusive, then the probability of the set of explanations is the sum of probabilities of each explanation.
    
    [Mutual Exclusion assumption]
Explanation of an answer: At a high level, the set of \( msw \)'s used in a derivation of the answer.

The probability of an explanation is the product of the probabilities of random variables in the explanation.

- If the \( msw \)'s in a derivation are all independent, then the probability of the explanation can be computed without materializing it.
  
  [Independence assumption]

The probability of an answer is the probability of the set of explanations of the answer.

- If explanations are pairwise mutually exclusive, then the probability of the set of explanations is the sum of probabilities of each explanation.
  
  [Mutual Exclusion assumption]

- If the set of explanations is finite, then this sum can be effectively computed.
  
  [Finiteness assumption]
PRISM’s inference procedure uses the Independence, Mutual Exclusion and Finiteness assumptions to compute probabilities of answers without materializing the explanations.
Generalizations

- PRISM’s inference procedure uses the Independence, Mutual Exclusion and Finiteness assumptions to compute probabilities of answers without materializing the explanations.
  - Inference mimics the best known algorithms for certain statistical models (e.g. Viterbi alg. for HMMs).
Generalizations

- PRISM’s inference procedure uses the Independence, Mutual Exclusion and Finiteness assumptions to compute probabilities of answers without materializing the explanations.
  - Inference mimics the best known algorithms for certain statistical models (e.g. Viterbi alg. for HMMs).
- ProbLog and PITA (an implementation of LPAD) use BDDs to represent the set of explanations, and consequently remove Independence and Mutual Exclusion assumptions.
Generalizations

- PRISM’s inference procedure uses the Independence, Mutual Exclusion and Finiteness assumptions to compute probabilities of answers without materializing the explanations.
  - Inference mimics the best known algorithms for certain statistical models (e.g. Viterbi alg. for HMMs).
- ProbLog and PITA (an implementation of LPAD) use BDDs to represent the set of explanations, and consequently remove Independence and Mutual Exclusion assumptions.
  - Finiteness assumption is still needed since the BDDs need to be effectively constructed.
Probabilistic Systems

- **System Definitions**: Markov Chains (discrete- and continuous-time), Markov Decision Processes, Probabilistic Automata, recursive versions of some of the above, . . .
- **Property Specifications**: PCTL, PCTL*, CSL, GPL, . . .
- **Systems**: Prism, PreMo, UPPAAL-SMC, . . .

Systems have stochastic behavior

. . . in contrast to *Statistical Model Checking* where statistical (sampling) techniques are used to infer properties of non-probabilistic systems (with confidence bounds).
Probabilistic Transition Systems in PRISM

Example Markov Chain

\[
\begin{array}{c}
S_0 \\
S_1 \\
S_2 \\
S_3 \\
S_4
\end{array}
\]

- \( S_0 \) to \( S_1 \) with probability 0.4
- \( S_1 \) to \( S_0 \) with probability 0.3
- \( S_1 \) to \( S_2 \) with probability 0.2
- \( S_2 \) to \( S_1 \) with probability 0.5
- \( S_3 \) to \( S_0 \) with probability 0.1
- \( S_3 \) to \( S_1 \) with probability 0.5
- \( S_4 \) to \( S_3 \) with probability 1

% Encoding as a Probabilistic LP

\[ \text{trans}(S, I, T) :- \text{msw}(t(S), I, T) . \]
Probabilistic Transition Systems in PRISM

Example Markov Chain

% Encoding as a Probabilistic LP
trans(S, I, T) :- msw(t(S), I, T).

% Ranges
:- values(t(s0), [s0, s1, s2]).
:- values(t(s1), [s1, s3, s4]).
:- values(t(s4), [s3]).

% Distributions
set_sw(t(s0), [0.5, 0.3, 0.2]).
set_sw(t(s1), [0.4, 0.1, 0.5]).
set_sw(t(s4), [1]).
Probabilistic Transition Systems in PRISM

Example Markov Chain

% Encoding as a Probabilistic LP
trans(S, I, T) :- msw(t(S), I, T).

% Encoding of Reachability
reach(S, I, T) :-
    trans(S, I, U),
    reach(U, next(I), T).
reach(S, _, S).
What is the probability of reaching $s_3$ via some path starting at $s_0$?

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

reach($S$, $I$, $T$) :-
    trans($S$, $I$, $U$),
    reach($U$, next($I$), $T$).
reach($S$, _, $S$).
What is the probability of reaching \( s_3 \) via some path starting at \( s_0 \)?

\[ \text{?- prob}(	ext{reach}(s_0, 0, s_3)). \]

### Probabilistic Tabled Logic Programming

- `trans(S, I, T) :- msw(t(S), I, T).`
- `reach(S, I, T) :- trans(S, I, U), reach(U, next(I), T).`
- `reach(S, _, S).`
What is the probability of reaching \( s_3 \) via some path starting at \( s_0 \)?

\[
|?- \text{prob}(\text{reach}(s_0, 0, s_3)).
\]

Evaluation of the above query will not terminate!

\[
\text{trans}(S, I, T) :- \\
\quad \text{msw}(t(S), I, T).
\]

\[
\text{reach}(S, I, T) :- \\
\quad \text{trans}(S, I, U), \\
\quad \text{reach}(U, \text{next}(I), T). \\
\quad \text{reach}(S, _, S).
\]
Probabilistic Model Checking as Query Evaluation

What is the probability of reaching $s_3$ via some path starting at $s_0$?

$\text{?- } \text{prob}((\text{reach}(s_0, 0, s_3)))$.

Evaluation of the above query will not terminate!

There are infinitely many explanations for $\text{reach}(s_0, 0, s_3)$.

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

reach($S$, $I$, $T$) :-
  trans($S$, $I$, $U$),
  reach($U$, next($I$), $T$).

reach($S$, _, $S$).
What is the probability of reaching $s_3$ via some path starting at $s_0$?

 Evaluation of the above query will not terminate!

 There are infinitely many explanations for $\text{reach}(s_0, 0, s_3)$

 Distribution semantics is well-defined and gives the correct probability, but
What is the probability of reaching $s_3$ via some path starting at $s_0$?

?- prob(reach($s_0$, 0, $s_3$)).

Evaluation of the above query will not terminate!

There are infinitely many explanations for $\text{reach}(s_0, 0, s_3)$

Distribution semantics is well-defined and gives the correct probability, but

PRISM/ProbLog/PITA cannot evaluate this query.
What is the probability of reaching $s_3$ via some path starting at $s_0$?

?- prob(reach(s0, 0, s3)).

Evaluation of the above query will not terminate!

There are infinitely many explanations for reach(s0, 0, s3)

Distribution semantics is well-defined and gives the correct probability, but

PRISM/ProbLog/PITA cannot evaluate this query.

“PIP” solves this problem [Gorlin, R. & Smolka, ICLP’12].
Explanations for `reach(s0,0,s3)`: 

- `msw(t(s0), 0, s1), msw(t(s1), next(0), s3).`
- `msw(t(s0), 0, s0), msw(t(s0), next(0), s1), msw(t(s1), next(next(0)), s3).`
- `msw(t(s0), 0, s1), msw(t(s1), next(0), s1), msw(t(s1), next(next(0)), s3).`
- ...
Explanations

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

reach(S, I, T) :-
    trans(S, I, U),
    trans(U, next(I), T),
    reach(U, next(I), T).
reach(S, _, S).

Note: prob(reach(s0, 0, s3)) is same as prob(reach(s0, H, s3)) for any H.
Explanations

Note: prob(reach(s0,0,s3)) is same as prob(reach(s0,H,s3)) for any H.

We can use a grammar to represent the set of explanations for the abstracted query.

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

reach(S, I, T) :-
    trans(S, I, U),
    reach(U, next(I), T).
reach(S, _, S).

C. R. Ramakrishnan
Probabilistic Tabled Logic Programming
Explanations

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

reach(S, I, T) :-
    trans(S, I, U),
    reach(U, next(I), T).
reach(S, _, S).

Note: prob(reach(s0, 0, s3)) is same as prob(reach(s0, H, s3)) for any H.

We can use a grammar to represent the set of explanations for the abstracted query.

expl(reach(s0, H, s3)) →
    [msw(t(s0), H, s0)],
    expl(reach(s0, next(H), s3)).
expl(reach(s0, H, s3)) →
    [msw(t(s0), H, s1)],
    expl(reach(s1, next(H), s3)).
Explanations

\begin{align*}
\text{trans}(S, I, T) & : - \\
& \text{msw}(t(S), I, T).
\end{align*}

\begin{align*}
\text{reach}(S, I, T) & : - \\
& \text{trans}(S, I, U), \\
& \text{reach}(U, \text{next}(I), T).
\end{align*}

\begin{align*}
\text{reach}(S, _, S).
\end{align*}

expl(reach(s0, H, s3)) \rightarrow \\
[\text{msw}(t(s0), H, s0)], \\
expl(reach(s0, \text{next}(H), s3)).

expl(reach(s0, H, s3)) \rightarrow \\
[\text{msw}(t(s0), H, s1)], \\
expl(reach(s1, \text{next}(H), s3)).

\text{is similar to the stochastic grammar:}

\begin{align*}
S_0 & \xrightarrow{0.5} S_0 \\
S_0 & \xrightarrow{0.3} S_1
\end{align*}

\text{whose probability is given by the least solution to the equation:}

\begin{align*}
x_0 & = 0.5x_0 + 0.3x_1
\end{align*}
A probabilistic logic program with annotations of the form `temporal(p/n − i)`.

Example: `temporal(reach/3−2)`
- `reach` is a `temporal` predicate
- The second argument of an atom with root `reach` is its `instance argument`.

For a rule defining a temporal predicate, the instance argument of the head must be a subterm of instance arguments of every temporal body predicate.

Example: `reach(S, I, T) :-
    trans(S, I, U),
    reach(U, next(I), T).`

Instance arguments are not bound to non-instance arguments, or vice versa.
Temporally Well-Formed Programs

- A probabilistic logic program with annotations of the form `temporal(p/n-i)`.
  - Example: `temporal(reach/3-2)`
  - `reach` is a `temporal` predicate
  - The second argument of an atom with root `reach` is its `instance argument`.

- For a rule defining a temporal predicate, the instance argument of the head must be a subterm of instance arguments of every temporal body predicate.
  - Example: `reach(S, I, T) :- trans(S, I, U), reach(U, next(I), T).

- Instance arguments are not bound to non-instance arguments, or vice versa.

- In explanation grammars of temporally well-formed programs, `msw(r, t, x)` will always be independent of any `msw` derived from non-terminal `expl(p)` if `t` is a proper subterm of `p`'s instance argument.
Factored Equation Diagrams

Not all explanation grammars can be translated directly to stochastic grammars.

Consider the explanation grammar for query
reach(s0, H, s3); reach(s0, H, s4).

The grammar will have productions of the form:
\[ \text{expl}(\text{reach}(s0, H, s3); \text{reach}(s0, H, s4)) \rightarrow \text{expl}(\text{reach}(s0, H, s3)). \]
\[ \text{expl}(\text{reach}(s0, H, s3); \text{reach}(s0, H, s4)) \rightarrow \text{expl}(\text{reach}(s0, H, s4)). \]

We can factor such grammars using Factored Explanation Diagrams (FEDs), which are similar to BDDs.
Structure of FEDs

FED is a labeled DAG with

- `tt` and `ff` as leaf nodes
- `msw(r, h)` is an `n`-ary node if `r` is a random process with `n` possible outcomes; outgoing edges are labeled with the outcomes.
- `expl(t, h)` is a binary node; outgoing edges are labeled 0 and 1.
- If there is an edge from `x_1` to `x_2`, then `x_1 < x_2` via a specially defined partial order relation.
Operations on FEDs

Boolean operations “∧” and “∨” can be performed on FEDs along the same line as on BDDs, with one significant change:

- BDD operations are based on a total node order.
- We only have a partial node order for FEDs.
- When we recursively push operations down the diagram, we may encounter incomparable nodes.
- We then generate a placeholder merge node, and process merges separately.
Operations on FEDs

Boolean operations “∧” and “∨” can be performed on FEDs along the same line as on BDDs, with one significant change:

- BDD operations are based on a *total* node order.
- We only have a partial node order for FEDs.
- When we recursively push operations down the diagram, we may encounter *incomparable* nodes.
- We then generate a placeholder *merge* node, and process merges separately.
- Note that *msw* nodes are always comparable; so a merge will involve at least one *expl* node.
- We expand (one of) the *expl* node(s) with its definition, and perform the postponed operation.
The probability of a set of explanations is computed by generating and solving a set of equations from its FED.

**FED for** \( \text{expl}(\text{reach}(s_0, s_3), H) \):  

\[
\begin{align*}
& \text{msw}(t(s_0), H) \\
& \text{expl}(\text{reach}(s_0, s_3), \text{next}(H)) \\
& \quad \text{expl}(\text{reach}(s_1, s_3), \text{next}(H))
\end{align*}
\]

\[
\begin{align*}
x_0 &= t_{00} \cdot x_0 + t_{01} \cdot x_1 \\
t_{00} &= 0.5 \\
t_{01} &= 0.3
\end{align*}
\]

**FED for** \( \text{expl}(\text{reach}(s_1, s_3), H) \):

\[
\begin{align*}
& \text{msw}(t(s_1), H) \\
& \text{expl}(\text{reach}(s_1, s_3), \text{next}(H)) \\
& \quad \text{expl}(\text{reach}(s_3, s_3), \text{next}(H)) \\
& \quad \quad \text{expl}(\text{reach}(s_4, s_3), \text{next}(H))
\end{align*}
\]

\[
\begin{align*}
x_1 &= t_{11} \cdot x_1 + t_{13} \cdot x_3 + t_{14} \cdot x_4 \\
t_{11} &= 0.4 \\
t_{13} &= 0.1 \\
t_{14} &= 0.5
\end{align*}
\]
FEDs to Equations

The probability of a set of explanations is computed by generating and solving a set of equations from its FED.

**FED for expl(reach(s0,s3), H):**

\[
\begin{align*}
\text{msw}(t(s0), H) \\
\text{expl(reach(s0,s3), next(H))} \\
\text{expl(reach(s1,s3), next(H))}
\end{align*}
\]

\[
\begin{align*}
x_0 &= t_{00} \times x_0 \\
&\quad + t_{01} \times x_1 \\
t_{00} &= 0.5 \\
t_{01} &= 0.3
\end{align*}
\]

**FED for expl(reach(s1,s3), H):**

\[
\begin{align*}
\text{msw}(t(s1), H) \\
\text{expl(reach(s1,s3), next(H))} \\
\text{expl(reach(s3,s3), expl(reach(s4,s3), expl(reach(s1,s3), next(H))))}
\end{align*}
\]

\[
\begin{align*}
x_1 &= t_{11} \times x_1 \\
&\quad + t_{13} \times x_3 \\
&\quad + t_{14} \times x_4 \\
t_{11} &= 0.4 \\
t_{13} &= 0.1 \\
t_{14} &= 0.5
\end{align*}
\]

The least solution to these monotone polynomial equations gives the probability of the set of explanations.
Probabilistic Computation Tree Logic (PCTL)

- PCTL is a logic for specifying properties of Probabilistic Transition Systems (Discrete-Time Markov Chains), where a subset of predefined *propositions*, $A$, hold at states.
- State formulas, $\varphi$, defined over individual states:
  
  $$
  A \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \Pr(\psi) > b \mid \Pr(\psi) \geq b
  $$

- Path formulas, $\psi$, defined over execution paths:
  
  $$
  \phi_1 U \phi_2 \mid X \phi
  $$

- State formulas are non-probabilistic; path formulas have associated probabilities.
- Used as the property specification language by many systems, including the Prism Model Checker.
% State Formulae
models(S, prop(A)) :-
    holds(S, A).
models(S, neg(SF)) :-
    not models(S, SF).
models(S, and(SF1, SF2)) :-
    models(S, SF1),
    models(S, SF2).
models(S, pr(PF, gt, B)) :-
    prob(pmodels(S, PF), P),
    P > B.
models(S, pr(PF, geq, B)) :-
    prob(pmodels(S, PF), P),
    P >= B.

% Path Formulae
pmodels(S, PF) :-
    pmodels(S, PF, _).
:- table pmodels/3.
pmodels(S, until(SF1, SF2), H) :-
    models(S, SF2).
pmodels(S, until(SF1, SF2), H) :-
    models(S, SF1),
    trans(S, H, T),
    pmodels(T, until(SF1, SF2), next(H)).
pmodels(S, next(SF), H) :-
    trans(S, H, T),
    models(T, SF).
temporal(pmodels/3-3).
Prototype: PCTL Model Checking

5 processes:

- Time performance is compared with that of the Prism Model Checker.
- System specified using Prism’s modeling language (Reactive Modules, RM). Markov Chain derived from direct logical encoding of the semantics of RM.

6 processes:

- Chosen benchmark:
  - System: Synchronous Leader Election protocol
  - Property: “eventually a leader is elected” (reachability).
- Model checking times are within a factor of 3 (note log scale).
Reactive Probabilistic Labeled Transition Systems (RPLTS)

- Automata has finite number of states.
- Each state offers a finite number of actions, each with a distinct label.
- 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]
Generalized Probabilistic Logic (GPL)

[Cleaveland, Iyer & Narasimha, TCS’05]

- An expressive, mu-calculus-based, logic for branching-time probabilistic processes.
- Strictly more expressive than PCTL*.
- Can be used to construct model checkers for recursive Markov Chains.
- *Thus far, no model checker was available*!!
- We can construct a model checker for GPL by directly encoding its semantics as a probabilistic logic program.
Usual mu-calculus-like modalities and fixed points (called “state formulae”) in GPL.

Fuzzy formulae, $\psi$, have a probabilistic interpretation: each formula’s truth value has a probability associated with it.

$$\psi = \psi \lor \psi \mid \psi \land \psi \mid \langle a \rangle \psi \mid [a] \psi \mid \phi \mid X$$

State formulae, $\phi$, have a boolean interpretation:

$$\phi = \phi \lor \phi \mid \cdots \mid \text{pr}^{B} \psi \mid \text{pr}^{B} \psi \mid \cdots \text{ propositions} \cdots$$

Alternation-free fixed point equations of the form $X =_{\mu} \psi$ and $X =_{\nu} \psi$. 
### GPL Model Checker

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>Prolog Code</th>
</tr>
</thead>
</table>
| \%
| `pmodels(S, PF, H): S is in the model of fuzzy formula PF at or after instant H` |
| \%
| `smodels(S, SF): S is in the model of state formula SF` |
| `pmodels(S, sf(SF), H) :-
| smodels(S, SF).` |
| `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).` |
| `pmodels(S, diam(A, F), H) :-
| action(S, A, SW),
| msw(SW, H, T),
| pmodels(T, F, [T,SW|H]).` |
| `pmodels(S, box(A, F), H) :-
| findall(SW, action(S,A,SW), L),
| all_pmodels(L, S, F, H).` |
| `pmodels(S, form(X), H) :-
| tabled_pmodels(S, X, H1), H=H1.` |
| `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).` |
| `:- table tabled_pmodels/3.` |
| `tabled_pmodels(S,X,H) :-
| fdef(X, lfp(F)),
| pmodels(S, F, H).` |
Recursive Markov Chains (RMCs)

Markov chains with *calls* and *returns* [Etessami & Yannakakis, 2005, ...]

- Probabilistic Push-Down Systems [Kucera, Esparza & Mayr, 2006]
- PreMo system [Wojtczak & Etessami, 2008]
Reachability in RMCs

Transform into a Reactive Probabilistic LTS:

- Labels on probabilistic transitions are all $p$ (omitted in figure).
- Check reachability using the following GPL formula:

$X_i$: eventually exit $ex_i$ is reached:

$$X_i = \mu \langle e_i \rangle tt \lor \langle p \rangle X_i$$
$$\lor (\langle c \rangle X_1 \land \langle r_1 \rangle X_i)$$
$$\lor (\langle c \rangle X_2 \land \langle r_2 \rangle X_i)$$
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 in PRISM, we introduce \( \text{nd}(X, I, V) \) 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: \( \text{nd}(s_2, 0, X) \) with \( \text{values}(s_2, [b, c]) \) 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.)

\[
q(Y) :\leftarrow \text{nd}(f, 0, X), \\
\quad \text{msw}(X, 0, Y).
\]
\[
\text{values}(f, [a,b]). \\
\text{values}(a, [t,f]). \\
\text{values}(b, [t,f]). \\
\text{set\_sw}(a, [0.3, 0.7]) \\
\text{set\_sw}(b, [0.6, 0.4])
\]

?- \text{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.
Approximate Inference

Current, Preliminary Work, on MCMC-based Sampling

- Monte-Chain Monte Carlo: walk though the possible worlds.
- Gibbs sampler: walk by resampling one of the random variables in the current state.
- In our case, we consider a set of possible worlds as a state in the Markov Chain. Naive method:
  - Generate a sample derivation. Its msws define a set of possible worlds.
  - Choose an msw and resample; find a derivation consistent with the new set of possible worlds.
  - The set of msws in the new derivation forms the next state in the chain.
- Using explanations instead of derivations makes this method more complex ([Moldovan et al, ECSQARU’13])
Approximate Inference for Conditional Queries

- Naive method: use Metropolis-Hastings and reject samples inconsistent with evidence.
- Better methods: Adapt sampling to not generate inconsistent examples in the first place.
  - Adapt $\text{msw}$ distributions to minimize generation of samples inconsistent with evidence [e.g. Mansinghka ’09].
  - Adapt the Markov Chain based on prior rejections to focus on consistent part of the state space [classical adaptive MCMC].
Current and Future Work

Sampling-Based Inference
Structure Learning (ILP)

Decision Support / Planning
Statistical Model Checking

Different Forms of Uncertainty
Expectations
“Stratification”
Co-Authors

- Samik Basu
- Yifei Dong
- Vic Du
- Andrey Gorlin
- Md. Asiful Islam
- Narayan Kumar
- Giri Pemmasani
- Bob Pokorny
- Arun Nampally
- I. V. Ramakrishnan
- Y. S. Ramakrishna
- Abhik Roychoudhury
- Dipti Saha
- Beata Sarna-Starosta
- Anu Singh
- Scott Smolka
- Scott Stoller
- Terry Swift
- David Warren
- Ping Yang