CHR(PRISM)-based Probabilistic Logic Learning

Jon Sneyers, Wannes Meert, Joost Vennekens
Dept. of Computer Science, K.U.Leuven, Belgium

Yoshitaka Kameya and Taisuke Sato
Tokyo Institute of Technology, Japan

ICLP 2010, Edinburgh, Scotland, UK, July 2010
1. CH RiSM
   - Introduction
   - CH RiSM
   - Examples
   - PRISM features in CH RiSM

2. Semantics and Ambiguity

3. Implementation of CH RiSM

4. Related work and Conclusion
High-level language *extension*

- different host languages (originally and mostly Prolog)
  - e.g. CHR(Prolog), CHR(Haskell), CHR(Java), CHR(C)

- Multi-headed committed-choice guarded rewrite rules

- Originally: designed for writing constraint solvers

- Today: general-purpose programming language
Declarative semantics:
- classical logic semantics
- linear logic semantics [Betz & Frühwirth 2005]

Abstract operational semantics $\omega_a$

Theoretical operational semantics $\omega_t$
- adds propagation history to avoid trivial nontermination

Refined operational semantics $\omega_r$ [Duck et al. 2004]
- activate constraints depth-first, left-to-right
- search for matching rules by trying occurrences in textual order

Priority semantics $\omega_p$ [De Koninck et al. 2007]
- apply rules in order of priority
- also dynamic priorities
CHiRiSM is a probabilistic variant of CHR
- based on CHR(PRISM)
- PRISM: PRogramming In Statistical Modeling
  [Sato 1995, Sato & Kameya 1997]
- CHiRiSM: CHance Rules induce Statistical Models
Operational semantics as usual \((\omega_t, \omega_r, \omega_p)\)

New in CHRI SM:
- rules have a probability \(P\) (default is \(P = 1\))
- rule instances are considered:
  - when a rule is considered, a biased coin is tossed
  - with probability \(P\), the rule is applied
  - with probability \(1 - P\), the rule is not applied
  - each rule instance can only be considered once
- probabilistic disjunctions in the body:
  - one disjunct is randomly chosen (committed-choice)

Also, PRISM is the host language so its builtins can be used
Example 1: generating random graphs

CHRiSM program

0.5 ?? node(A), node(B) ==> edge(A,B).

Example interaction

| ?- sample node(a),node(b),node(c)  
node(a),node(b),node(c) <=>  
node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a).  
| ?- sample node(a),node(b),node(c)  
node(a),node(b),node(c) <=>
node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a).
Example 1: generating random graphs

CHRiSM program

0.5 ?? node(A), node(B) ==> edge(A,B).

Example interaction

| ?- sample node(a),node(b),node(c) 
node(a),node(b),node(c) <==>
node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a).
| ?- sample node(a),node(b),node(c) 
node(a),node(b),node(c) <==>
node(c),node(b),node(a),edge(a,c),edge(b,c),edge(c,a),
edge(c,b),edge(a,b).
Example 1: generating random graphs

CHRIISM program

0.5 ?? node(A), node(B) ==> edge(A,B).

Example interaction

| ?- sample node(a),node(b),node(c) node(a),node(b),node(c) <==>
node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a).
| ?- sample node(a),node(b),node(c) node(a),node(b),node(c) <==>
node(c),node(b),node(a),edge(a,c),edge(b,c),edge(c,a),
edge(c,b),edge(a,b).
Example 1: generating random graphs

**CHRiSM program**

0.5 ?? node(A), node(B) ==> edge(A,B).

**Example interaction**

\[
\begin{align*}
| & \text{?- sample node(a),node(b),node(c)} \\
& \text{node(a),node(b),node(c) <=>} \\
& \text{node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a).} \\
| & \text{?- sample node(a),node(b),node(c)} \\
& \text{node(a),node(b),node(c) <=>} \\
& \text{node(c),node(b),node(a),edge(a,c),edge(b,c),edge(c,a),} \\
& \quad \text{edge(c,b),edge(a,b).}
\end{align*}
\]
Example 1: generating random graphs

**CHRiSM program**

0.5 ?? node(A), node(B) ==> edge(A,B).

**Example interaction**

| ?- sample node(a),node(b),node(c) node(a),node(b),node(c) <==> node(c),node(b),node(a),edge(b,c),edge(c,b),edge(b,a). |
| ?- sample node(a),node(b),node(c) node(a),node(b),node(c) <==> node(c),node(b),node(a),edge(a,c),edge(b,c),edge(c,a), edge(c,b),edge(a,b). |
Example 2: coin toss

CHRiSM program

toss <=> head:0.5 ; tail:0.5.

Example interaction

| ?- sample toss
toss <==> tail.
Example 2: coin toss

**CHRiSM program**

```plaintext
/toss <=> ?? head;tail.
```

**Example interaction**

```
| ?- sample toss
|   toss <=> tail.
| ?- sample toss,toss
|   toss,toss <=> head,tail.
| ?- prob toss,toss,toss,toss <==> head,head

Probability of toss,toss,toss <==> head,head is: 0.25
```
Example 2: coin toss

**CHRiSM program**

```
toss <=> ?? head;tail.
```

**Example interaction**

```
?- sample toss
  toss <=> tail.
?- sample toss,toss
  toss,toss <=> head,tail.
?- prob toss,toss <=> head,head
  Probability of toss,toss<==>head,head is: 0.25
```
Example 2: coin toss

CHRiSM program

\[
toss \iff \text{?? head;tail.}
\]

Example interaction

\[
| \text{- sample toss} \\
toss \iff \text{tail.} \\
| \text{- sample toss,toss} \\
toss,toss \iff \text{head,tail.} \\
| \text{- prob toss,toss \iff head,head} \\
\text{Probability of toss,toss \iff head,head is: 0.25}
\]
**Example 2: coin toss**

**CHRiSM program**

```christmas
\[\text{toss} \leftrightarrow \text{?? head;tail.}\]
```

**Example interaction**

```
\|- \text{sample toss}
\text{toss} \leftrightarrow \text{tail.}
\|- \text{sample toss,toss}
\text{toss,toss} \leftrightarrow \text{head, tail.}
\|- \text{prob toss,toss} \leftrightarrow \text{head,head}
Probability of \text{toss,toss} \leftrightarrow \text{head,head} is: 0.25
```
Example 2: coin toss

CHiRiSM program

\[
\text{toss }\leftrightarrow\ ??\ \text{head;tail.}
\]

Example interaction

\[
\begin{align*}
\text{?- sample toss} \\
\text{toss }\leftrightarrow\ \text{tail.} \\
\text{?- sample toss,toss} \\
\text{toss,toss }\leftrightarrow\ \text{head,tail.} \\
\text{?- prob toss,toss }\leftrightarrow\ \text{head,head} \\
\text{Probability of toss,toss }\leftrightarrow\ \text{head,head is: 0.25}
\end{align*}
\]
Example 2: coin toss

CHRiSM program

\[
\text{toss} \leftrightarrow \text{?? head;tail.}
\]

Example interaction

\[
\begin{align*}
&\text{?- sample toss} \\
&toss \leftrightarrow \text{tail.} \\
&\text{?- sample toss,toss} \\
&toss,toss \leftrightarrow \text{head,tail.} \\
&\text{?- prob toss,toss} \leftrightarrow \text{head,head} \\
\text{Probability of toss,toss} \leftrightarrow \text{head,head is: 0.25}
\end{align*}
\]
CHRiSM program

\[
toss \iff \text{?? head;tail.}
\]

Example interaction

\[
| \text{?- sample tossing} \\
| \text{toss } \iff \text{tail.} \\
| \text{?- sample tossing, tossing} \\
| \text{toss, tossing } \iff \text{head, tail.} \\
| \text{?- prob tossing, tossing } \iff \text{head, head} \\
\]
Probability of tossing, tossing \iff head, head is: 0.25
CHRiSM program

player(P) <=> c(P) ?? rock(P);scissors(P);paper(P).

rock(P1), scissors(P2) ==> winner(P1).
scissors(P1), paper(P2) ==> winner(P1).
paper(P1), rock(P2) ==> winner(P1).

Example interaction

?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),rock(tom).
?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),paper(tom),winner(tom).

Probability of player(tom),player(jon)===>winner(tom) is: 0.33333
Example 3: rock-scissors-paper

**CHRiSM program**

\[
\text{player(P)} \iff \text{c(P)} \oplus \text{rock(P)}; \text{scissors(P)}; \text{paper(P)}.
\]

\[
\text{rock}(P_1), \text{scissors}(P_2) \implies \text{winner}(P_1).
\]

\[
\text{scissors}(P_1), \text{paper}(P_2) \implies \text{winner}(P_1).
\]

\[
\text{paper}(P_1), \text{rock}(P_2) \implies \text{winner}(P_1).
\]

**Example interaction**

\[
| \text{?- sample player(tom),player(jon)}
\text{player(tom),player(jon)} \iff \text{rock}(jon),\text{rock}(tom).
\]

\[
| \text{?- sample player(tom),player(jon)}
\text{player(tom),player(jon)} \iff \text{rock}(jon),\text{paper}(tom),\text{winner}(tom).
\]

\[
| \text{?- prob player(tom),player(jon) ===> winner(tom)}
\text{Probability of player(tom),player(jon)===>winner(tom) is: 0.33333}
\]
Example 3: rock-scissors-paper

CHRiSM program

\[
\text{player(P)} \iff \text{c(P)} \ ?\? \text{rock(P)};\text{scissors(P)};\text{paper(P)}. \\
\text{rock(P1), scissors(P2)} \implies \text{winner(P1)}. \\
\text{scissors(P1), paper(P2)} \implies \text{winner(P1)}. \\
\text{paper(P1), rock(P2)} \implies \text{winner(P1)}. 
\]

Example interaction

\[
| \text{?- sample player(tom),player(jon)} \\
\text{player(tom),player(jon)} \iff \text{rock(jon),rock(tom)}. \\
| \text{?- sample player(tom),player(jon)} \\
\text{player(tom),player(jon)} \iff \text{rock(jon),paper(tom),winner(tom)}. \\
| \text{?- prob player(tom),player(jon) ===> winner(tom)} \\
\text{Probability of player(tom),player(jon)===>winner(tom) is: 0.33333} 
\]
**Example 3: rock-scissors-paper**

**CHRiSM program**

```
player(P) <-> c(P) ?? rock(P);scissors(P);paper(P).

rock(P1), scissors(P2) => winner(P1).
scissors(P1), paper(P2) => winner(P1).
paper(P1), rock(P2) => winner(P1).
```

**Example interaction**

```prolog
| ?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),rock(tom).
| ?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),paper(tom),winner(tom).
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.33333
```
Example 3: rock-scissors-paper

**CHRiSM program**

```prolog
player(P) <= c(P) ?? rock(P);scissors(P);paper(P).

rock(P1), scissors(P2) ==> winner(P1).
scissors(P1), paper(P2) ==> winner(P1).
paper(P1), rock(P2) ==> winner(P1).
```

**Example interaction**

```
| ?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),rock(tom).
| ?- sample player(tom),player(jon)
player(tom),player(jon) <==> rock(jon),paper(tom),winner(tom).
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon) ===> winner(tom) is: 0.33333
```
Example 3: rock-scissors-paper

**CHRiSM program**

```
player(P) <=> c(P) ?? rock(P);scissors(P);paper(P).

rock(P1), scissors(P2) ==> winner(P1).
scissors(P1), paper(P2) ==> winner(P1).
paper(P1), rock(P2) ==> winner(P1).
```

**Example interaction**

```
| ?- sample player(tom),player(jon)
player(tom),player(jon) <=> rock(jon),rock(tom).
| ?- sample player(tom),player(jon)
player(tom),player(jon) <=> rock(jon),paper(tom),winner(tom).
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.33333
```
Example 3: rock-scissors-paper

CHiRiSM program

\[
\text{player}(P) \iff c(P) \quad ?? \quad \text{rock}(P) ; \text{scissors}(P) ; \text{paper}(P).
\]

\[
\text{rock}(P1), \text{scissors}(P2) \implies \text{winner}(P1).
\]

\[
\text{scissors}(P1), \text{paper}(P2) \implies \text{winner}(P1).
\]

\[
\text{paper}(P1), \text{rock}(P2) \implies \text{winner}(P1).
\]

Example interaction

\[
\text{?- sample player(tom),player(jon)}
\]

\[
\text{player(tom),player(jon) } \iff \text{rock(jon),rock(tom)}.
\]

\[
\text{?- sample player(tom),player(jon)}
\]

\[
\text{player(tom),player(jon) } \iff \text{rock(jon),paper(tom),winner(tom)}.
\]

\[
\text{?- prob player(tom),player(jon) } \implies \text{winner(tom)}
\]

Probability of player(tom),player(jon) ===> winner(tom) is: 0.33333
Features of PRISM

- PRISM has many nice features, a.o.:
  - Probabilistic execution (sample)
  - Probability computation (prob)
  - EM-learning (learn)
- These features can also be used in CHRiSM
- Probabilistic execution: `sample goal`
  - starting from goal, apply CHiRiSM rules

- Probability computation: `prob goal <==> result`
  - compute probability that "sample goal" gives "result" (full observation)
  - `prob goal ===> result`
    - compute probability that "sample goal" gives something of the form "result, otherstuff" (partial observation)

- EM-learning: `learn(observations)`
  - observations: a list of observations of the form "goal <==> result" or "goal ===> result"
  - compute an assignment to the unknown probabilities such that the likelihood of the observations is maximized
Example: learning

**CHRiSM program**

```
player(P) <=> P ? rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
...```

**Example interaction**

```
| ?- learn([ (50 times player(tom),player(jon) ===> winner(tom)),
             (20 times player(tom),player(jon) ===> winner(jon)),
             (30 times player(tom),player(jon) ===> ¬winner(tom),¬winner(jon))])
...
| ?- show_sw
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.499604
```
**Example: learning**

**CHRiSM program**

```
player(P) <-> P ?? rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
```

**Example interaction**

```
| ?- learn([ (50 times player(tom),player(jon) ==> winner(tom)),
             (20 times player(tom),player(jon) ==> winner(jon)),
             (30 times player(tom),player(jon) ==> ~winner(tom),~winner(jon))])
#goals: 0(3)
Exporting switch information to the EM routine ... done
#em-iterations: 0..(23) (Converged: -102.965335828)
Statistics on learning:
Graph size: 72
Number of switches: 2
Number of switch instances: 6
Number of iterations: 23
Final log likelihood: -102.965335828
Total learning time: 0.000 seconds
Explanation search time: 0.000 seconds
Total table space used: 40496 bytes
Type show_sw or show_sw_b to show the probability distributions....
| ?- show_sw
```

Jon Sneyers et.al.  CHR(PRISM)-based Probabilistic Logic Learning
**Example: learning**

**CHRiSM program**

```prolog
player(P) <=> P ?? rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
...```

**Example interaction**

```prolog
| ?- learn([ (50 times player(tom),player(jon) ===> winner(tom)),
             (20 times player(tom),player(jon) ===> winner(jon)),
             (30 times player(tom),player(jon) ===> ~winner(tom),~winner(jon))])
...
| ?- show_sw
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon) ===> winner(tom) is: 0.499604```
**Example: learning**

**CHiRiSM program**

```
player(P) <=> P ?? rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
...
```

**Example interaction**

```
| ?- learn([ (50 times player(tom),player(jon) ==> winner(tom)),
            (20 times player(tom),player(jon) ==> winner(jon)),
            (30 times player(tom),player(jon) ==> ~winner(tom),~winner(jon))])
...
| ?- show_sw
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)
| ?- prob player(tom),player(jon) ==> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.499604
```
**Example: learning**

**CHRiSM program**

```prolog
player(P) <=> P ?? rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
...
```

**Example interaction**

```prolog
| ?- learn([ (50 times player(tom),player(jon) ===> winner(tom)),
             (20 times player(tom),player(jon) ===> winner(jon)),
             (30 times player(tom),player(jon) ===> ~winner(tom),~winner(jon))])
...
| ?- show_sw
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.499604
```
CHRiSM program

\[ \text{player}(P) \iff P \text{ ?} \text{ rock}(P) \lor \text{ scissors}(P) \lor \text{ paper}(P). \]
\[ \text{rock}(P1), \text{scissors}(P2) \implies \text{winner}(P1). \]
...

Example interaction

\[ \text{?- learn}([ (50 \text{ times} \ \text{player}(\text{tom}), \text{player}(\text{jon}) \implies \text{winner}(\text{tom})),\]
\[ (20 \text{ times} \ \text{player}(\text{tom}), \text{player}(\text{jon}) \implies \text{winner}(\text{jon})),\]
\[ (30 \text{ times} \ \text{player}(\text{tom}), \text{player}(\text{jon}) \implies \neg \text{winner}(\text{tom}), \neg \text{winner}(\text{jon}))]) \]
...

\[ \text{?- show_sw} \]
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)

\[ \text{?- prob player(tom), player(jon) \implies \text{winner}(tom)} \]
Probability of player(tom),player(jon) \implies \text{winner}(tom) is: 0.499604
**Example: learning**

**CHRiSM program**

```
player(P) <=> P ?= rock(P) ; scissors(P) ; paper(P).
rock(P1), scissors(P2) ==> winner(P1).
...
```

**Example interaction**

```
| ?- learn([ (50 times player(tom),player(jon) ===> winner(tom)),
            (20 times player(tom),player(jon) ===> winner(jon)),
            (30 times player(tom),player(jon) ===> ~winner(tom),~winner(jon))])
| ...
| ?- show_sw
Switch exp1(jon): 1 (p: 0.60057034) 2 (p: 0.06536821) 3 (p: 0.33406143)
Switch exp1(tom): 1 (p: 0.08420895) 2 (p: 0.20973622) 3 (p: 0.70605482)
| ?- prob player(tom),player(jon) ===> winner(tom)
Probability of player(tom),player(jon)===>winner(tom) is: 0.499604
```
Chance rules (may) have two kinds of probabilities:
- Rule: application probability
- Body: probabilistic disjunction

Syntax: rule with probability Prob

```
Prob ?? Head <=> Guard | Body.
```

default: “1 ??” (normal CHR rule)

Syntax: probabilistic disjunction (in rule body)

fixed probability distribution: (cf. CP-Logic [Vennekens et al. 2006])

```
D1:Prob1 ; D2:Prob2 ; ... ; DN:ProbN
```

unknown probability distribution:

```
Prob ?? D1 ; D2 ; ... ; DN
```
Chance rules (may) have two kinds of probabilities:
- Rule: application probability
- Body: probabilistic disjunction

**Syntax: rule with probability Prob**

```
Prob ?? Head <=> Guard | Body.
```

**default:** “1 ??” (normal CHR rule)

**Syntax: probabilistic disjunction (in rule body)**

Fixed probability distribution: (cf. CP-Logic [Vennekens et al. 2006])
```
D1:Prob1 ; D2:Prob2 ; ... ; DN:ProbN
```

Unknown probability distribution:
```
Prob ?? D1 ; D2 ; ... ; DN
```
 syntax of chance rules

- Chance rules (may) have two kinds of probabilities:
  - Rule: application probability
  - Body: probabilistic disjunction

**Syntax: rule with probability Prob**

```
Prob ?? Head <=> Guard | Body.
```

default: “1 ??” (normal CHR rule)

**Syntax: probabilistic disjunction (in rule body)**

- fixed probability distribution: (cf. CP-Logic [Vennekens et al. 2006])
  
  ```
  D1:Prob1 ; D2:Prob2 ; ... ; DN:ProbN
  ```

- unknown probability distribution:
  
  ```
  Prob ?? D1 ; D2 ; ... ; DN
  ```
Different kinds of probability expressions Prob allowed:

- numbers:
  \[ \text{head:0.5 ; tail:0.5} \]

- arithmetic expression which is ground at runtime:
  \[ \text{eval}(1-K) ?? \text{maybe\_keep\_me}(K) \leftrightarrow \text{true}. \]

- probabilities are unknown:
  \[ \text{roll} \leftrightarrow ?? 1 ; 2 ; 3 ; 4 ; 5 ; 6 \]

- probabilities are unknown and parametrized:
  \[ \text{roll}(\text{Die}) \leftrightarrow \text{Die} ?? 1 ; 2 ; 3 ; 4 ; 5 ; 6 \]

- Numbers and arithmetic expressions: fixed probabilities
- Parametrized probabilities (with 0 or more parameters):
  - initially: uniform distribution
  - actual distribution can be learned from examples
Formal operational semantics: see paper

Distribution semantics:

- Given a program and a query,
- there are many possible results, each with some probability
- the probability may depend on the execution strategy (ambiguity)
- If the program is unambiguous, the distribution semantics is well-defined
- Details: see paper
Example:

0.5 ?? a <=> b.
0.5 ?? a <=> c.

What is the result of “\texttt{prob a <=> b}”? 
- If the execution strategy considers rule 1 first, then 50% 
- If the execution strategy considers rule 2 first, then 25%

The example program is \textbf{ambiguous} w.r.t. the general strategy class $\Omega_t$
- However, it is not ambiguous w.r.t. the refined strategy class $\Omega_r$
- Under the refined semantics, rule 1 is always considered first
1. CHRiSM

2. Semantics and Ambiguity

3. Implementation of CHRiSM
   - PRISM
   - CHR(PRISM)
   - Result-directed execution

4. Related work and Conclusion
- PRISM extends Prolog with probabilistic built-ins
- Implemented on top of B-Prolog [Zhou 1994-2009]
- Also several CHR systems available for B-Prolog
First prototype used toychr [Duck 2004]
  - naive implementation of CHR, very inefficient
  - uses only pure Prolog

Current implementation is based on the Leuven CHR system [Schrijvers and Demoen 2004]
  - efficient implementation of CHR
  - uses “dirty” Prolog

Translation CHriSM $\rightarrow$ CHR(PRISM) is more or less straightforward

Some details in the paper
Result-directed execution

- For normal execution (sample), everything is OK
- For explanation search (prob and learn), normal execution is too naive
- Can be solved, see CHR workshop
1. CHRI$$\text{SM}$$
2. Semantics and Ambiguity
3. Implementation of CHRI$$\text{SM}$$
4. Related work and Conclusion
   - PCHR
   - Other related formalisms
   - Conclusion
Another probabilistic variant of CHR: PCHR

- Probabilistic CHR: rules get a weight/probability
- Coin toss in PCHR:

PCHR program

\begin{verbatim}
toss <=>0.5: head.
toss <=>0.5: tail.
\end{verbatim}

- Semantics: $\omega_t$ semantics, where the applied rule is probabilistically chosen from all applicable rules
- Probability distribution given by rule weights: probability $= \text{normalized weight}$
Problems with PCHR

- Only simplification rules
- Not easy to see probability of a rule:
  - depends on its weight and weights of all other applicable rules
  - meaning of a rule is non-local: depends on the entire program
- PCHR instantiates $\omega_t$, but its semantics depends on the full non-determinism of $\omega_t$
  - ‘refined’ or ‘priority’ PCHR is probably not desirable
  - efficient implementation of PCHR is not straightforward
Advantages of CHiRM over PCHR

- CHiRM semantics are more natural:
  - PCHR does not make sense for probabilistic propagation rules, CHiRM does
  - chance rules have a *local* meaning, PCHR rules don’t

- CHiRM semantics are more usable:
  - CHiRM allows more execution control ($\omega_r$ and $\omega_p$ semantics)
  - CHiRM should allow efficient implementation

- PRISM features can be used in CHiRM
  - PCHR has only sampling, no learning etc.
  - no need to reinvent the wheel
Other related formalisms

- CP-logic (LPADs) [Vennekens et al. 2006] can be encoded in CH RiSM
  - details in the CHR 2009 paper
- Many other probabilistic logic programming formalisms are sublogics of CP-logic:
  - PRISM itself
  - ProbLog [De Raedt et al. 2007]
  - ICL [Poole 1997]
- Bayesian network-inspired formalisms:
  - BLP [Kersting & De Raedt 2007] covered by CH RiSM
  - Others are more difficult (they support more complicated distributions):
    - RBN [Jaeger 1997]
    - CLP(BN) [Santos Costa et al. 2008]
    - Blog [Milch et al. 2007]
New way to add probabilities to CHR: CHRiSM
  - based on CHR(PRISM)
  - has advantages over PCHR

Efficient implementation:

Subsumes other probabilistic-logic formalisms

Exploratory work: still a lot to be done!
Future work

- Language design
  - current syntax/semantics good in practice?
  - need to investigate much more examples
  - perhaps consider CHR(ProbLog) too?

- Currently: only ground goals/results → extend to non-ground

- Notion of probabilistic termination
  - program can terminate probabilistically but not classically
  - no problem for sampling
  - problem (of PRISM) for probability computation / learning

- Applications
  - Ongoing work: APOPCALEAPS, a pop music generator and learner
Questions?