From PRISM to ProbLog: There and Back Again

Alexander Vandenbroucke and Tom Schrijvers

KU LEUVEN

Motivation

Motivation

	PRISM	ProbLog

Motivation

	PRISM	ProbLog
Horn Clauses	✓	✓

Motivation

	PRISM	ProbLog
Horn Clauses	✓	✓
Semantics	**Distribution Semantics**	**Distribution Semantics**

Motivation

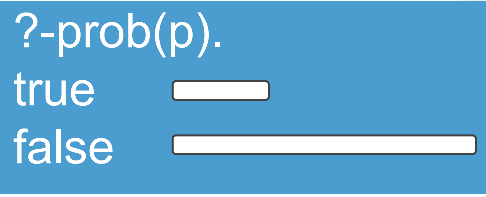
	PRISM	ProbLog
Horn Clauses	✓	✓
Semantics	**Distribution Semantics**	**Distribution Semantics**

	`values_x(i,[t,f],`
	`[0.5,0.5]).`
	`p :- msw(i,t),msw(i,t).`
		`0.5:msw(i,t) ; 0.5:msw(i,f)`
		
		`p :- msw(i,t),msw(i,t).`

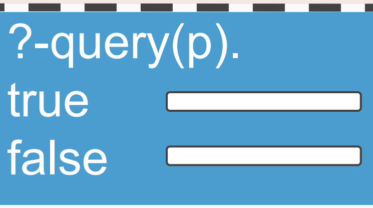
Motivation

	PRISM	ProbLog
Horn Clauses	✓	✓
Semantics	**Distribution Semantics**	**Distribution Semantics**

	`values_x(i,[t,f],`
	`           [0.5,0.5]).`
	`p :- msw(i,t),msw(i,t).`
		`0.5:msw(i,t) ; 0.5:msw(i,f)`
		
		`p :- msw(i,t),msw(i,t).`



```
?-prob(p).
```



```
?-query(p).
```

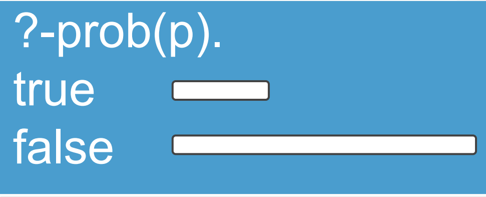
Motivation

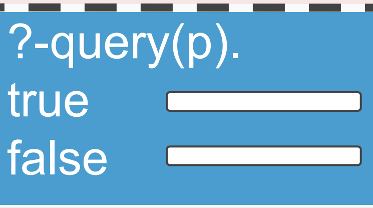
	PRISM	ProbLog
Horn Clauses	✓	✓
Semantics	**Distribution Semantics**	**Distribution Semantics**

```
values_x(i,[t,f],
         [0.5,0.5]).
p :- msw(i,t),msw(i,t).
```

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
```





No (stochastic) Memoisation

(stochastic) Memoisation

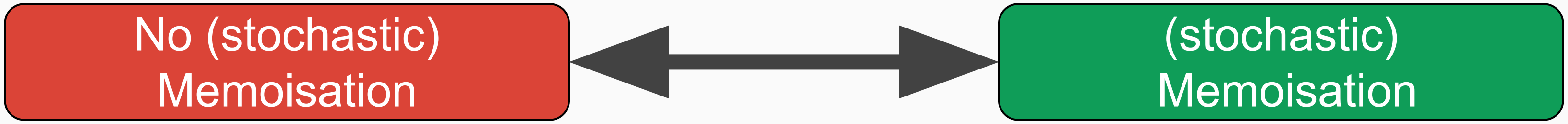
Motivation

No (stochastic) Memoisation

(stochastic) Memoisation

Is this difference fundamental?

Motivation



No (stochastic) Memoisation

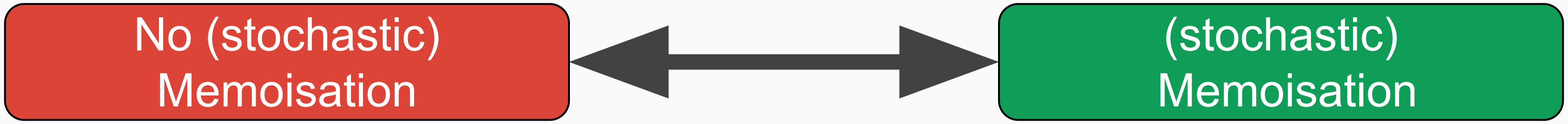
(stochastic) Memoisation

Is this difference fundamental?

Or

Can we transform one to the other and vice versa?

Motivation



No (stochastic) Memoisation

(stochastic) Memoisation

Is this difference fundamental?

Or

Can we transform one to the other and vice versa?

Yes

Semantics

Distribution Semantics

Program =

Probabilistic Facts + Logical Clauses

$\{ p_1 :: f_1, \ldots , p_n :: f_n \}$ p :- $q_1,\ldots,q_n$

F R

Sato, T.: A statistical learning method for logic programs with distribution semantics.
In: ICLP. pp. 715–729. MIT Press (1995)

Distribution Semantics

Total Choice C ⊆ F

$$P(C) =$$

Distribution Semantics

Total Choice C ⊆ F

$$P(C) = \prod_{f_i \in C} p_i$$

facts **true** in C

Distribution Semantics

Total Choice C ⊆ F

$$P(C) = \prod_{f_i \in C} p_i \times \prod_{f_i \notin C} (1-p_i)$$

facts **true** in C

facts **false** in C

Distribution Semantics

Total Choice C ⊆ F

$$P(C) = \prod_{f_i \in C} p_i \times \prod_{f_i \notin C} (1-p_i)$$

facts **true** in C

facts **false** in C

Probability of Query (atom) q

$$P_{F \cup R}(q) =$$

Distribution Semantics

Total Choice C ⊆ F

$$P(C) = \prod_{f_i \in C} p_i \times \prod_{f_i \notin C} (1-p_i)$$

facts **true** in C

facts **false** in C

Probability of Query (atom) q

$$P_{F \cup R}(q) = \sum_{\substack{C \subseteq F; \\ C \cup R \models q}} P(C)$$

all partial choices satisfying q

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
```

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
```

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
```

1 2

2 facts

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
```

1 1

1 fact

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
        1        2
```

2 facts
⇒ 4 possible worlds

1	2
T	T
T	F
F	T
F	F

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
        1        1
```

1 fact
⇒ 2 possible worlds

1
T
F

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
         1        2
```

2 facts

⇒ 4 possible worlds

1	2	p
T	T	T
T	F	F
F	T	F
F	F	F

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
         1        1
```

1 fact

⇒ 2 possible worlds

1	p
T	T
F	F

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
```

1 2

2 facts
⇒ 4 possible worlds

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
```

1 1

1 fact
⇒ 2 possible worlds

1	p	Pr
T	T	50%
F	F	50%

Distribution Semantics - Example

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
        1        2
```

2 facts
⇒ 4 possible worlds

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

```
0.5:msw(i,t) ; 0.5:msw(i,f)

p :- msw(i,t),msw(i,t).
        1        1
```

1 fact
⇒ 2 possible worlds

1	p	Pr
T	T	50%
F	F	50%

PRISM to ProbLog

PRISM to ProbLog

Labelling Each **Goal**

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
        1        2
```

```
0.5:msw(i,t) ; 0.5:msw(i,f).

p :- msw(i,t),msw(i,t).
        1         1
```

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

1	p	Pr
T	T	50%
F	F	50%

PRISM to ProbLog

Labelling Each **Goal**

```
values_x(i,[t,f],[0.5,0.5]).

p :- msw(i,t),msw(i,t).
         1         2
```

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)

p :- msw(i,t,g1),msw(i,t,g2).
         1            2
```

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

PRISM to ProbLog

Labelling Each **Clause**

```
values_x(i,[t,f],[0.5,0.5]).
p :- msw(i,X),q(X).
q(t).
q(f) :- msw(i,f).
```

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- msw(i,X,g1),q(X).
q(t).
q(f) :- msw(i,f,g1).
```

PRISM to ProbLog

Labelling Each **Clause**

```
values_x(i,[t,f],[0.5,0.5]).
p :- msw(i,X),q(X).
q(t).
q(f) :- msw(i,f).
```

1

2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- msw(i,X,g1),q(X).
q(t).
q(f) :- msw(i,f,g1).
```

1

1

1	2	p	q(f)	Pr
T	T	T	F	25%
T	F	T	T	25%
F	T	F	F	25%
F	F	T	T	25%

1	p	q(f)	Pr
T	T	F	50%
F	F	T	50%

PRISM to ProbLog

Labelling Each **Clause**

```
values_x(i,[t,f],[0.5,0.5]).
p :- msw(i,X),q(X).
q(t).
q(f) :- msw(i,f).
```

1

2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- msw(i,X,c1(g1)),q(X).
q(t).
q(f) :- msw(i,f,c3(g1)).
```

1

2

PRISM to ProbLog

Labelling Each **Clause**

```
values_x(i,[t,f],[0.5,0.5]).
p :- msw(i,X),q(X).
q(t).
q(f) :- msw(i,f).
```

1

2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- msw(i,X,c1(g1)),q(X).
q(t).
q(f) :- msw(i,f,c3(g1)).
```

1

2

1	2	p	q(f)	Pr
T	T	T	F	25%
T	F	T	T	25%
F	T	F	F	25%
F	F	T	T	25%

1	2	p	q(f)	Pr
T	T	T	F	25%
T	F	T	T	25%
F	T	F	F	25%
F	F	T	T	25%

PRISM to ProbLog

Labelling **Context**

```
values_x(i,[t,f],[0.5,0.5]).
p :- q,q.
q :- msw(i,t).
```

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- q,q.
q :- msw(i,t,c2(g1)).
```

PRISM to ProbLog

Labelling **Context**

```
values_x(i,[t,f],[0.5,0.5]).
p :- q,q.
q :- msw(i,t).
```

1 2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p :- q,q.
q :- msw(i,t,c2(g1)).
```

1 1

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

1	p	Pr
T	T	50%
F	F	50%

PRISM to ProbLog

Labelling **Context**

```
values_x(i,[t,f],[0.5,0.5]).
p :- q,q.
q :- msw(i,t).
```

1 2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p(C) :-
    q(c1(g1(C))),
    q(c1(g2(C))).
q(C) :- msw(i,t,c2(g1(C))).
```

1 2

PRISM to ProbLog

Labelling **Context**

```
values_x(i,[t,f],[0.5,0.5]).
p :- q,q.
q :- msw(i,t).
```

1 2

```
0.5:msw(i,t,_) ; 0.5:msw(i,f,_)
p(C) :-
    q(c1(g1(C))),
    q(c1(g2(C))).
q(C) :- msw(i,t,c2(g1(C))).
```

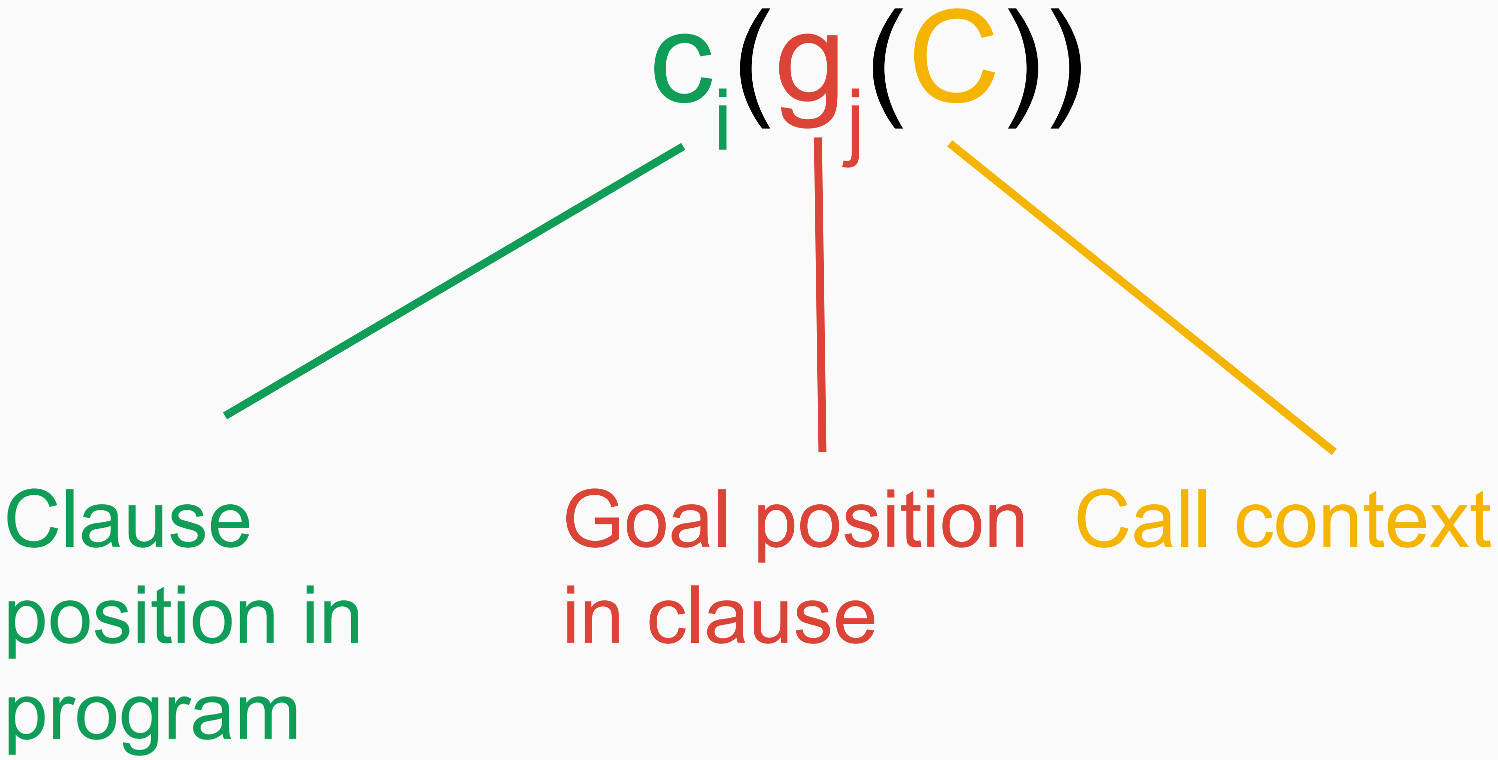
1 2

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

1	2	p	Pr
T	T	T	25%
T	F	F	25%
F	T	F	25%
F	F	F	25%

PRISM to ProbLog

Summary



$c_i(g_j(C))$

Clause position in program

Goal position in clause

Call context

PRISM to ProbLog

Summary

$$c_i(g_j(C))$$

⇒ Label traces SLD-resolution

ProbLog to PRISM

ProbLog to PRISM

Translate a fact

```
p :: fct
```

into

```
values_x(fct,[t,f],[p,1-p]).
fct :- msw(fct,X).
```

ProbLog to PRISM

Translate a fact

```
p :: fct
```

every fct is a **different** fact

into

```
values_x(fct,[t,f],[p,1-p]).
fct :- msw(fct,X).
```

ProbLog to PRISM

Translate a fact

```
p :: fct
```

into

```
values_x(fct,[t,f],[p,1-p]).
fct :- msw(fct,X).
```

ProbLog to PRISM

Assume

$$p_i :: f_1, \ldots , p_n :: f_n$$

is **finite**, and choose a value for each f_i up front.

ProbLog to PRISM

```
0.5 :: f1.
0.5 :: f2.
p :- f1, f2.
```

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.5,0.5]).
```

ProbLog to PRISM

```
0.5 :: f1.
0.5 :: f2.
p :- f1, f2.
```

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.5,0.5]).
p(Fs) :- f1(Fs), f2(Fs).
f1(Fs) :- member(f1,Fs).
f2(Fs) :- member(f2,Fs).
```

Passing *total choice* along

ProbLog to PRISM

```
query(p).
```

input list

output list

```
query :-
   choose(f1,[],F1),
   choose(f2,F1,F2),
   p(F2).
```

choose/3 ***probabilistically*** decides to place f_i in the list

ProbLog to PRISM

Assume

$$p_i :: f_1, \ldots , p_n :: f_n$$

is **finite**, and choose a value for each f_i up front.

this works

ProbLog to PRISM

Assume

$$p_i :: f_1, \ldots, p_n :: f_n$$

is **finite**, and choose a value for each f_i **up front**.

this works, but...

ProbLog to PRISM

Assume

$$p_1 :: f_1,\ p_2 :: f_2,\ \dots$$

is **potentially infinite**, and choose a value for each f_i **dynamically**.

ProbLog to PRISM - Dynamically

ProbLog to PRISM - Dynamically

- Pass a **partial** choice: a fact is either true, false, or unknown

ProbLog to PRISM - Dynamically

- Pass a **partial** choice: a fact is either true, false, or unknown
- when **encountering an unknown fact**: (1) abort

  (2) choose a value and extend the partial choice

  (3) restart the computation

ProbLog to PRISM - Dynamically

- Pass a **partial** choice: a fact is either true, false, or unknown
- when **encountering an unknown fact**: (1) abort
  (2) choose a value and extend the partial choice
  (3) restart the computation
  (4) backtrack over (2) when needed

ProbLog to PRISM - Example

```
0.5 :: f1.
0.4 :: f2.
p :- f1.
p :- f2.
```

ProbLog to PRISM - Example

```
0.5 :: f1.
0.4 :: f2.
p :- f1.
p :- f2.
```

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
```

```
p :- f1.
p :- f2.
```

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
```

```
p :- f1.
p :- f2.
```

`true` and `false` test the truth value in the partial choice

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
```

```
p :- f1.
p :- f2.
```

`true` and `false` test the truth value in the partial choice

`throw/1` throws an exception

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
f2(Pc) :- true(f2,Pc), !.
f2(Pc) :- not(false(f2,Pc)),throw(unknown(f2)).
p :- f1.
p :- f2.
```

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
f2(Pc) :- true(f2,Pc), !.
f2(Pc) :- not(false(f2,Pc)),throw(unknown(f2)).
p(Pc) :- f1(Pc).
p(Pc) :- f2(Pc).
```

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
f2(Pc) :- true(f2,Pc), !.
f2(Pc) :- not(false(f2,Pc)),throw(unknown(f2)).
p(Pc) :- f1(Pc).
p(Pc) :- f2(Pc).
query(Pc) :-
  catch(once(p(Pc)),unknown(F),extend(F,Pc)).
```

`catch(Goal,Ball,Handler)`, calls Goal. If an exception is thrown, it is unified with `Ball` and `Handler` is called

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
f2(Pc) :- true(f2,Pc), !.
f2(Pc) :- not(false(f2,Pc)),throw(unknown(f2)).
p(Pc) :- f1(Pc).
p(Pc) :- f2(Pc).
query(Pc) :-
  catch(once(p(Pc)),unknown(F),extend(F,Pc)).
```

call p only once to avoid counting a partial choice twice. (Exclusiveness condition)

ProbLog to PRISM - Example

```
values_x(f1,[t,f],[0.5,0.5]).
values_x(f2,[t,f],[0.4,0.6]).
f1(Pc) :- true(f1,Pc), !.
f1(Pc) :- not(false(f1,Pc)),throw(unknown(f1)).
f2(Pc) :- true(f2,Pc), !.
f2(Pc) :- not(false(f2,Pc)),throw(unknown(f2)).
p(Pc) :- f1(Pc).
p(Pc) :- f2(Pc).
query(Pc) :-
  catch(once(p(Pc)),unknown(F),extend(F,Pc)).
extend(F,Pc) :-
    msw(F,V), extend_pc(F,V,Pc,ExtendedPc),
    query(ExtendedPc).
```

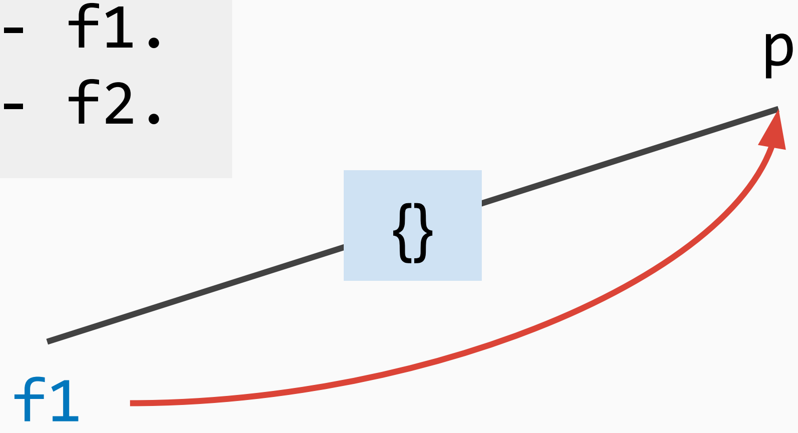
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```

p

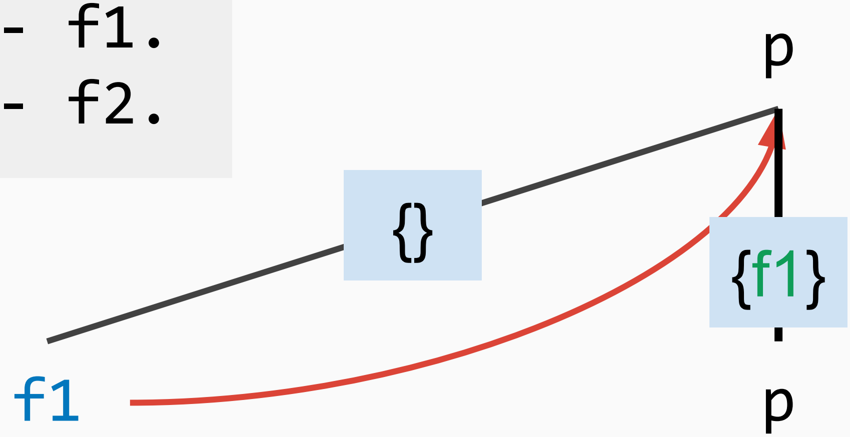
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



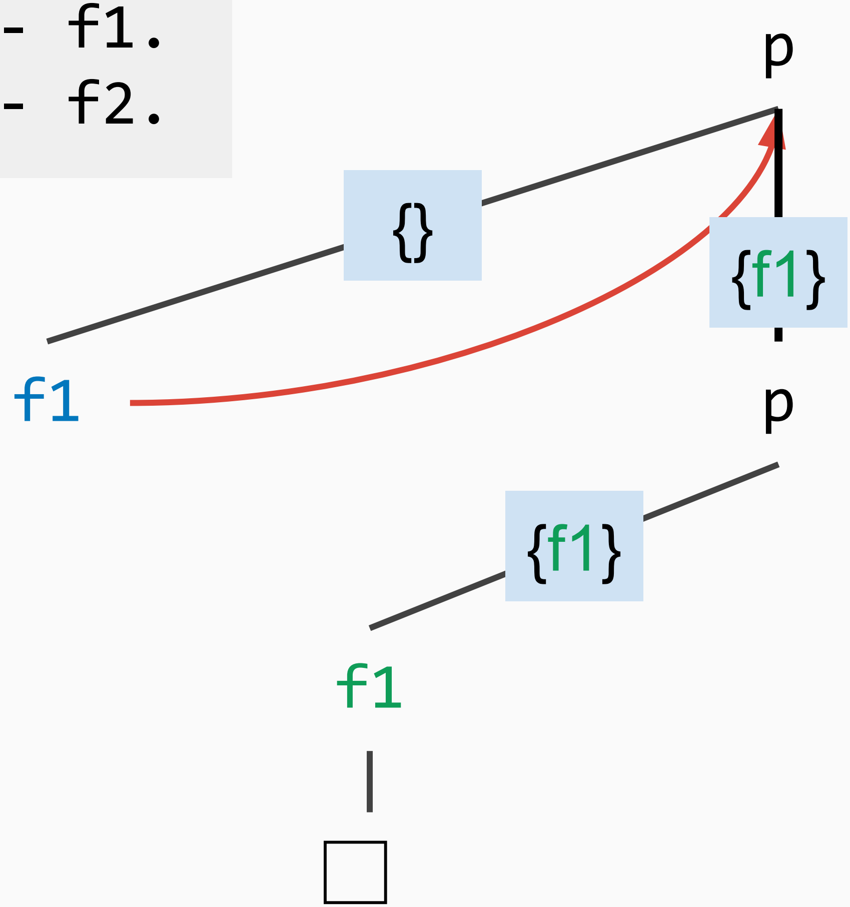
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



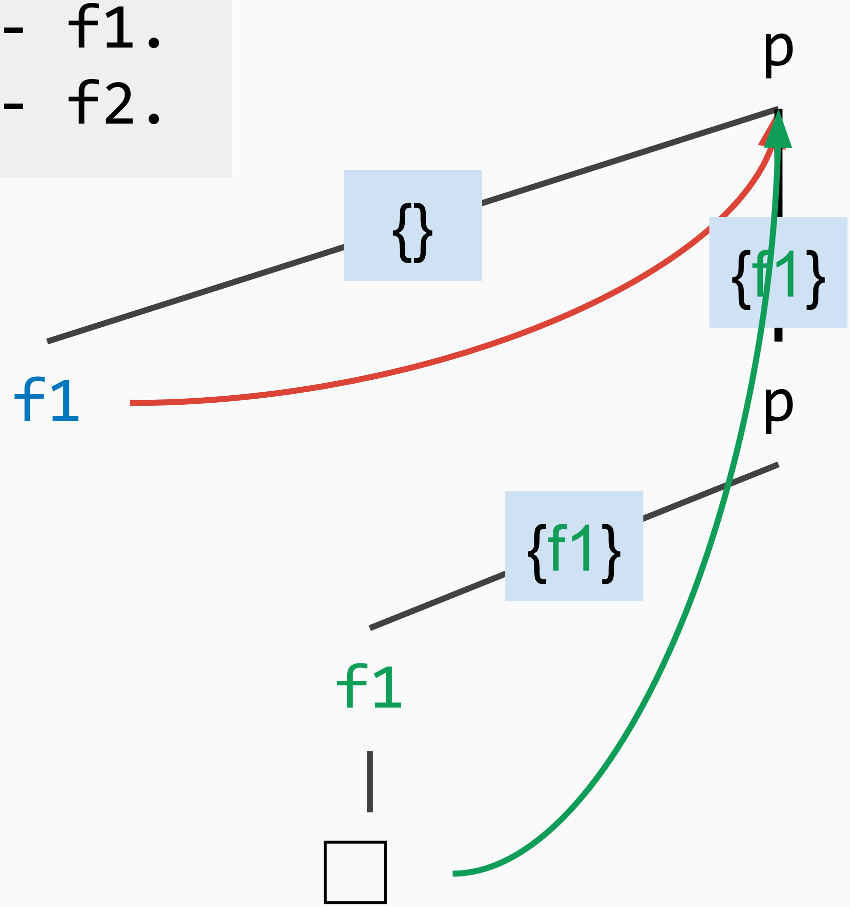
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



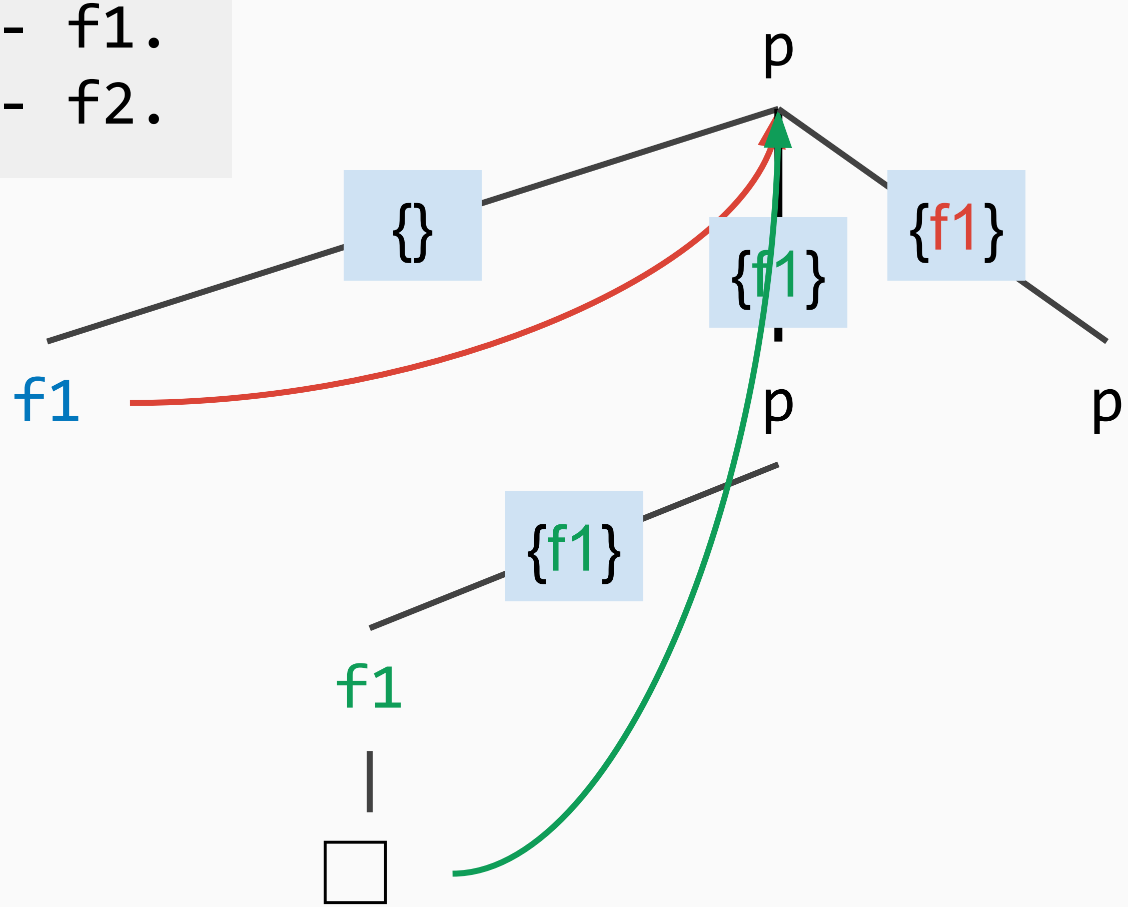
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



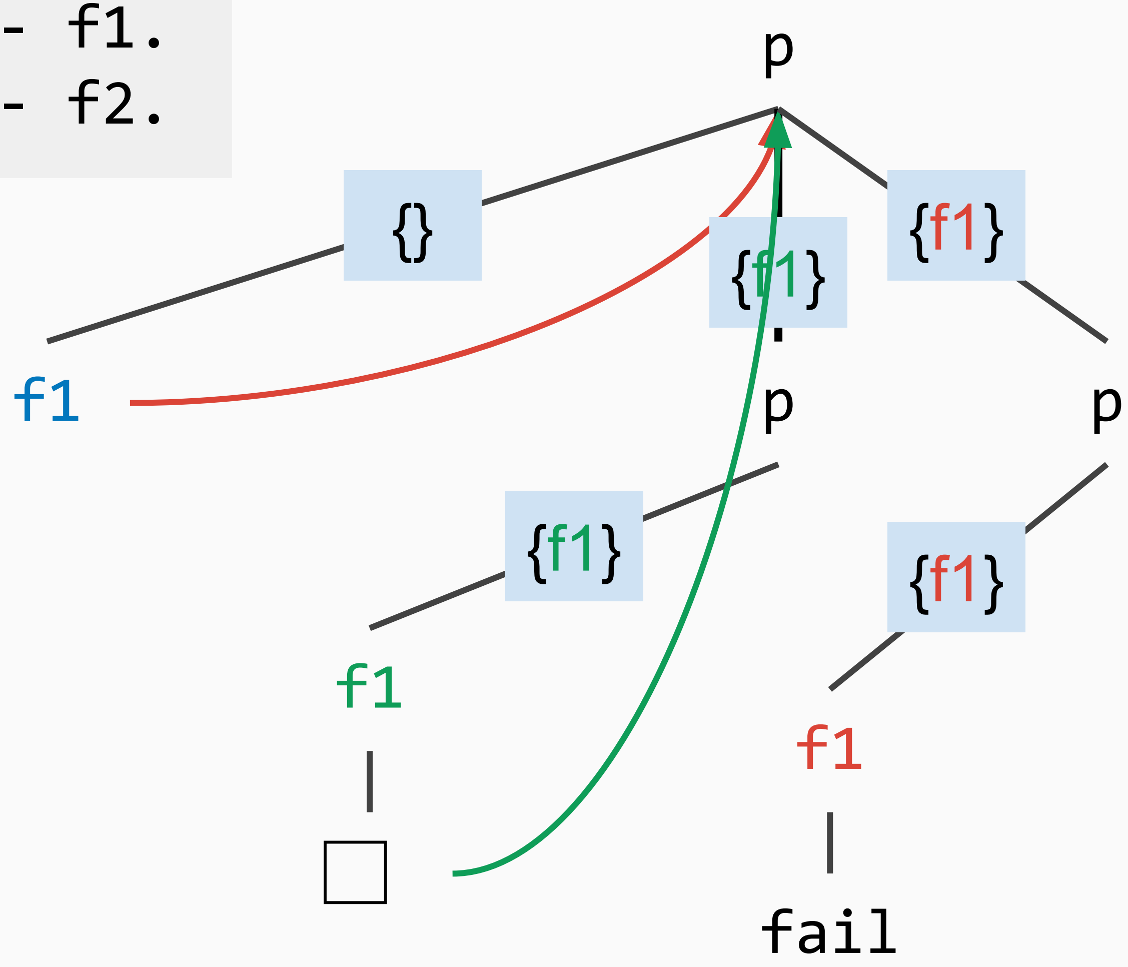
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



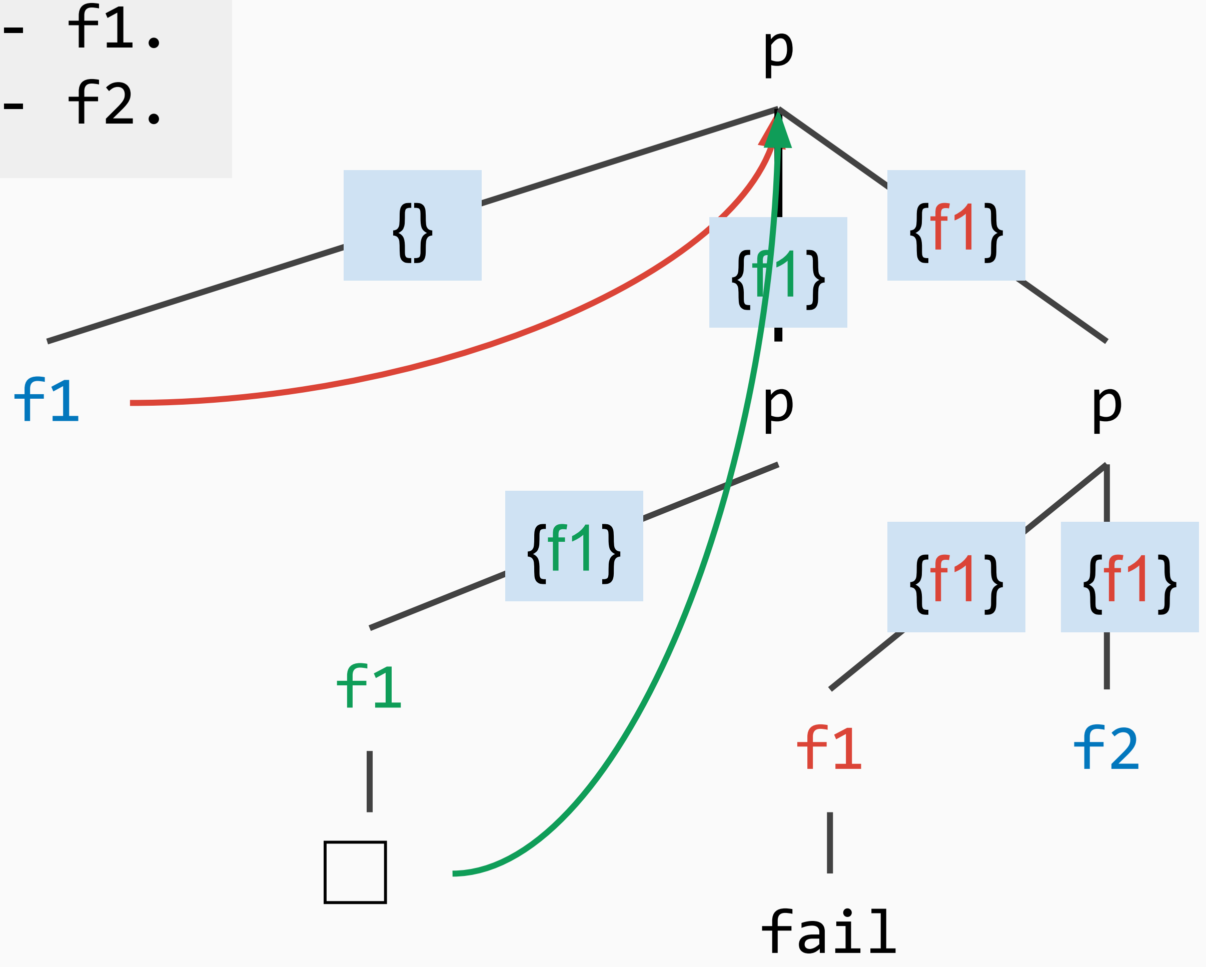
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



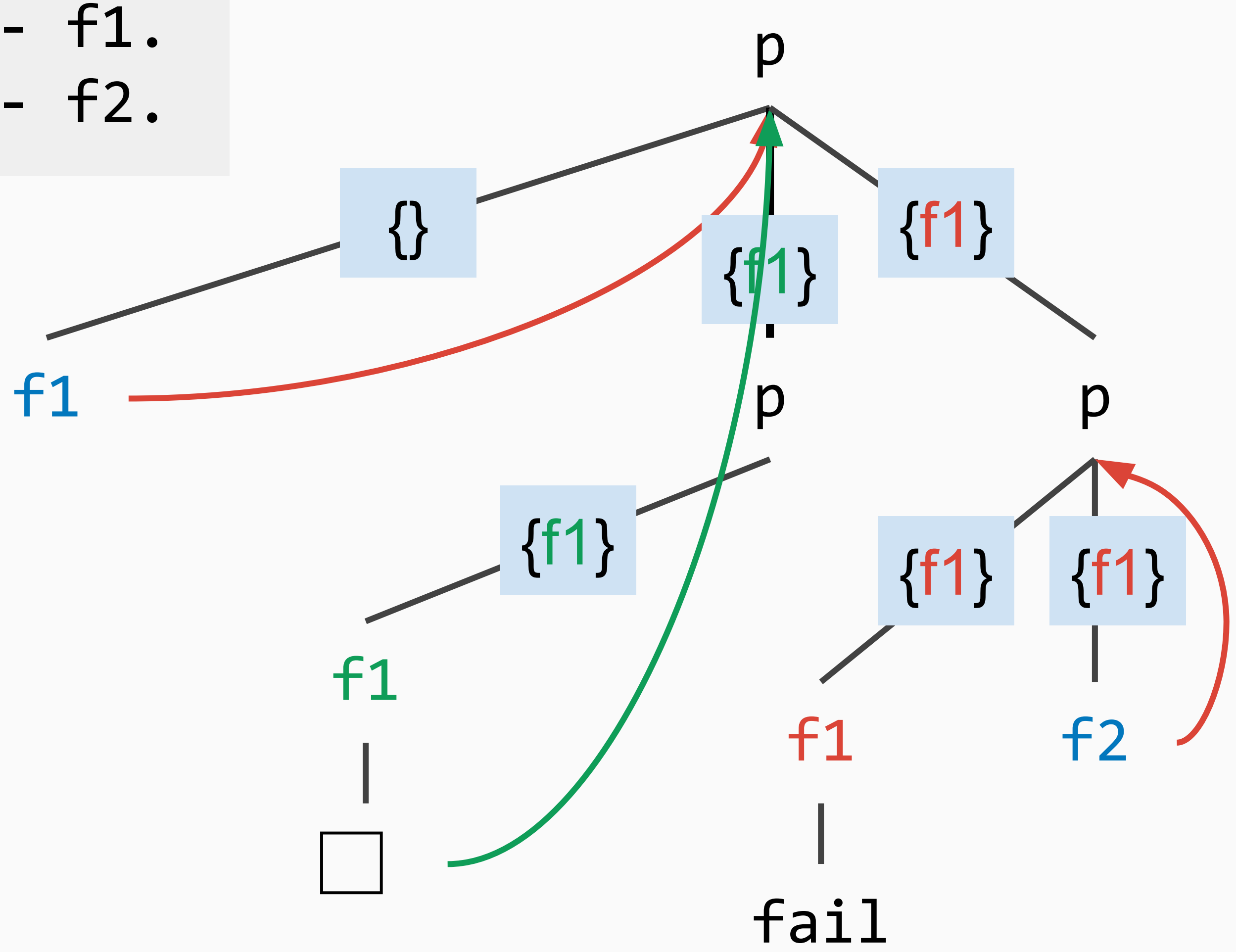
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



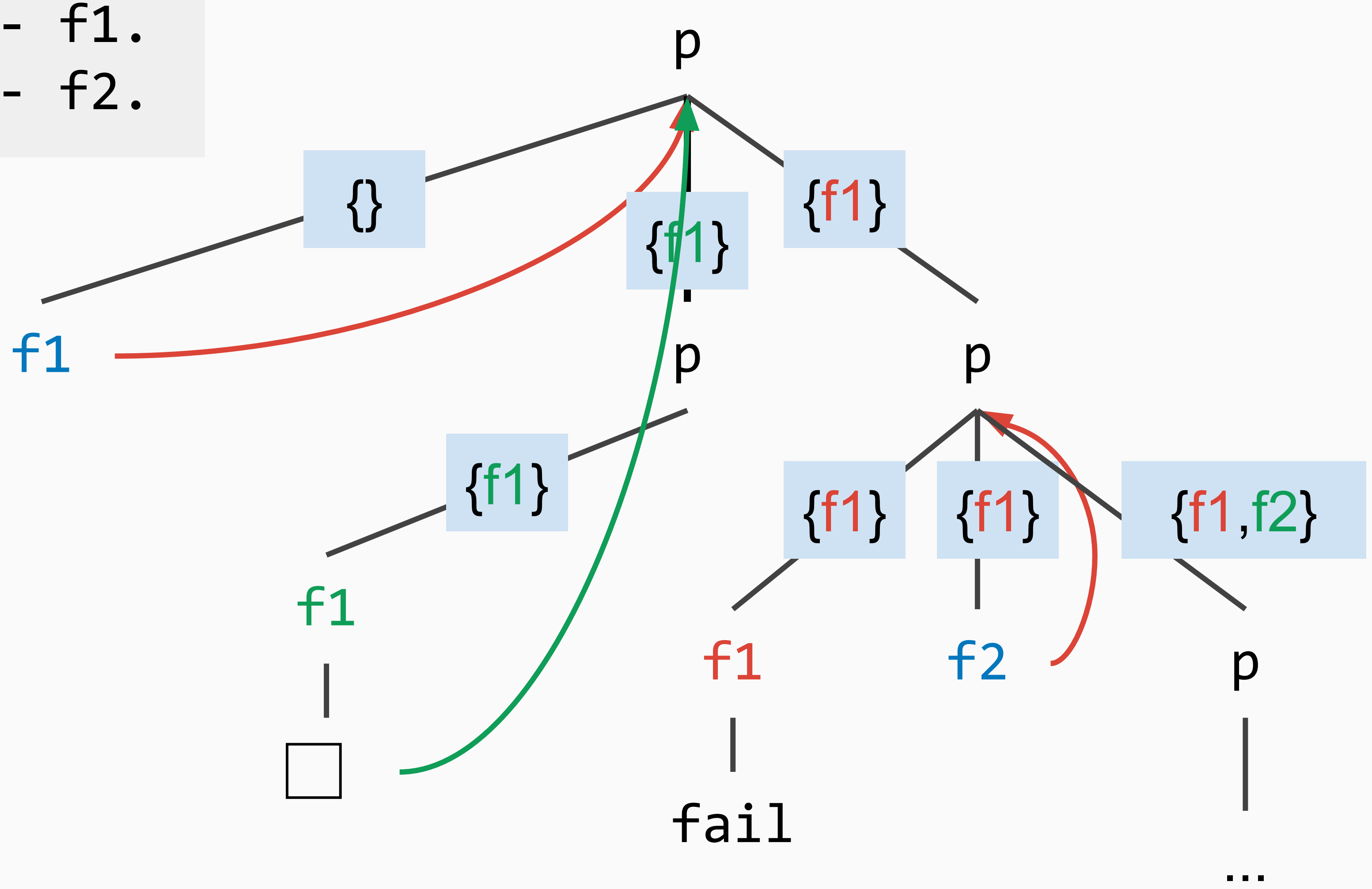
ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



ProbLog to PRISM - Execution Trace

```
p :- f1.
p :- f2.
```



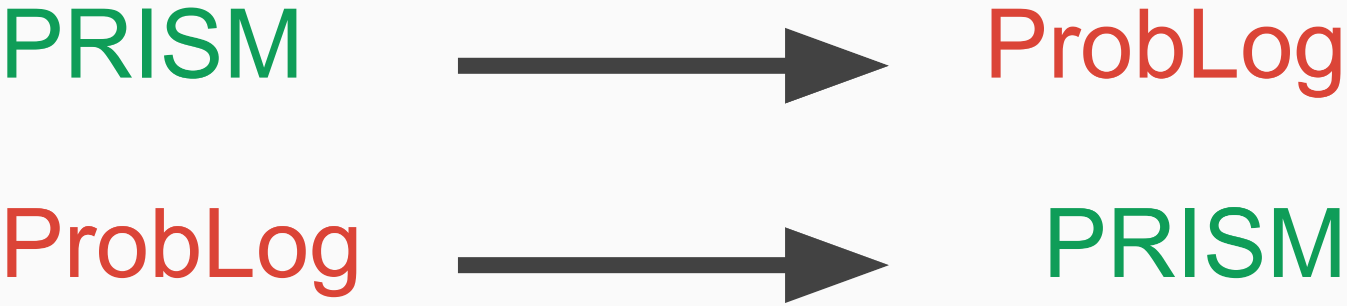
ProbLog to PRISM - Discussion

- ★ The transformation in the paper also deals with *facts with arguments*, and *flexible probabilities*.
- ★ It explores partial choices only *as far as necessary* to satisfy the query
- ★ It can still restart the program an exponential number of times in the worst case

Summary

Summary

2 transformations:

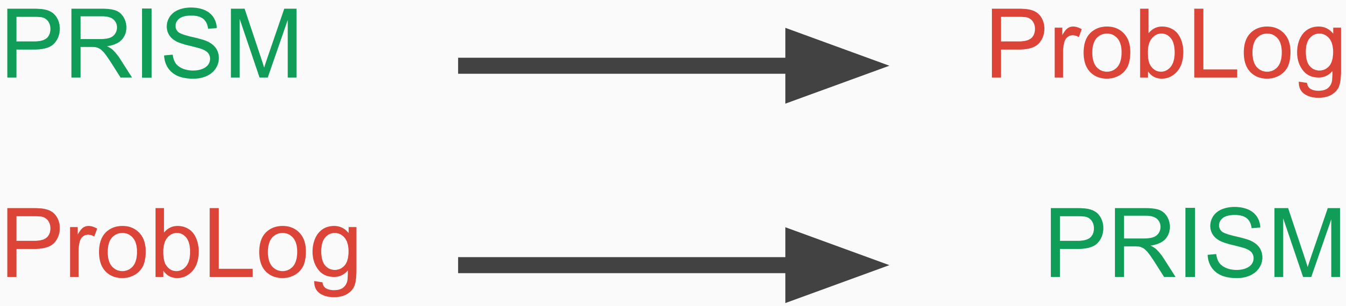


PRISM → ProbLog

ProbLog → PRISM

Summary

2 transformations:

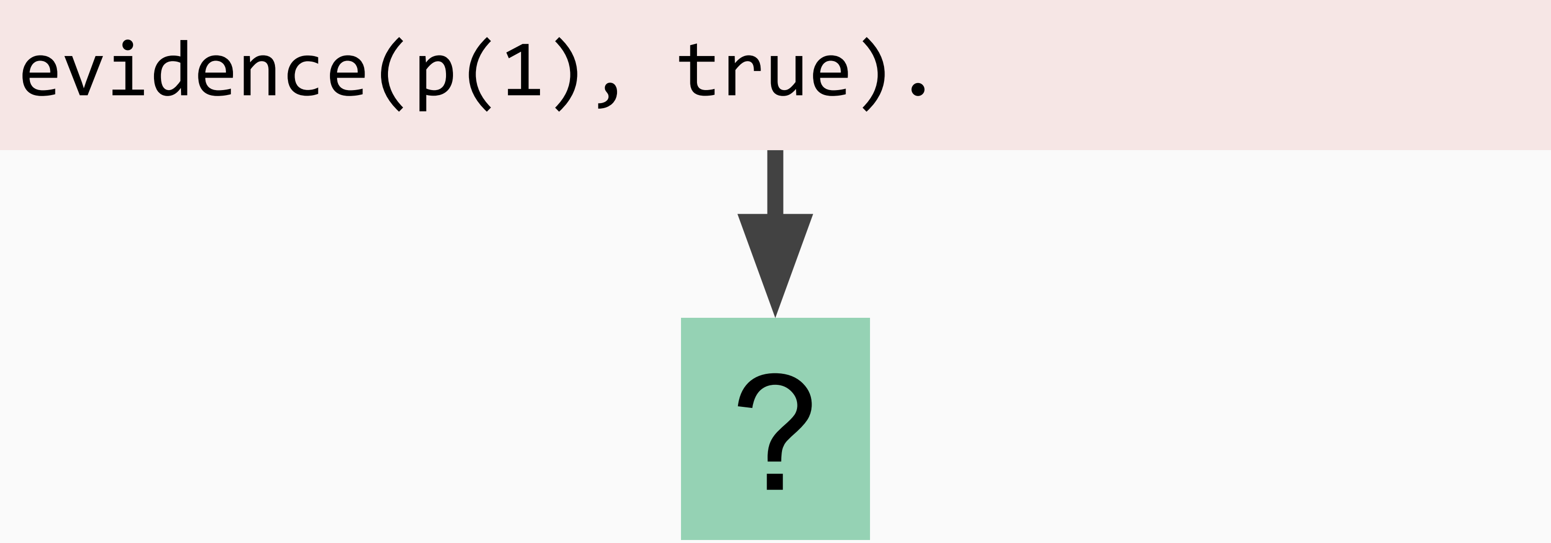


PRISM → ProbLog

ProbLog → PRISM

⇒ PRISM and ProbLog are not so different

Future Work - Observations



```
evidence(p(1), true).
```

Future Work - Correctness

When found, contact alexander.vandenbroucke@kuleuven.be

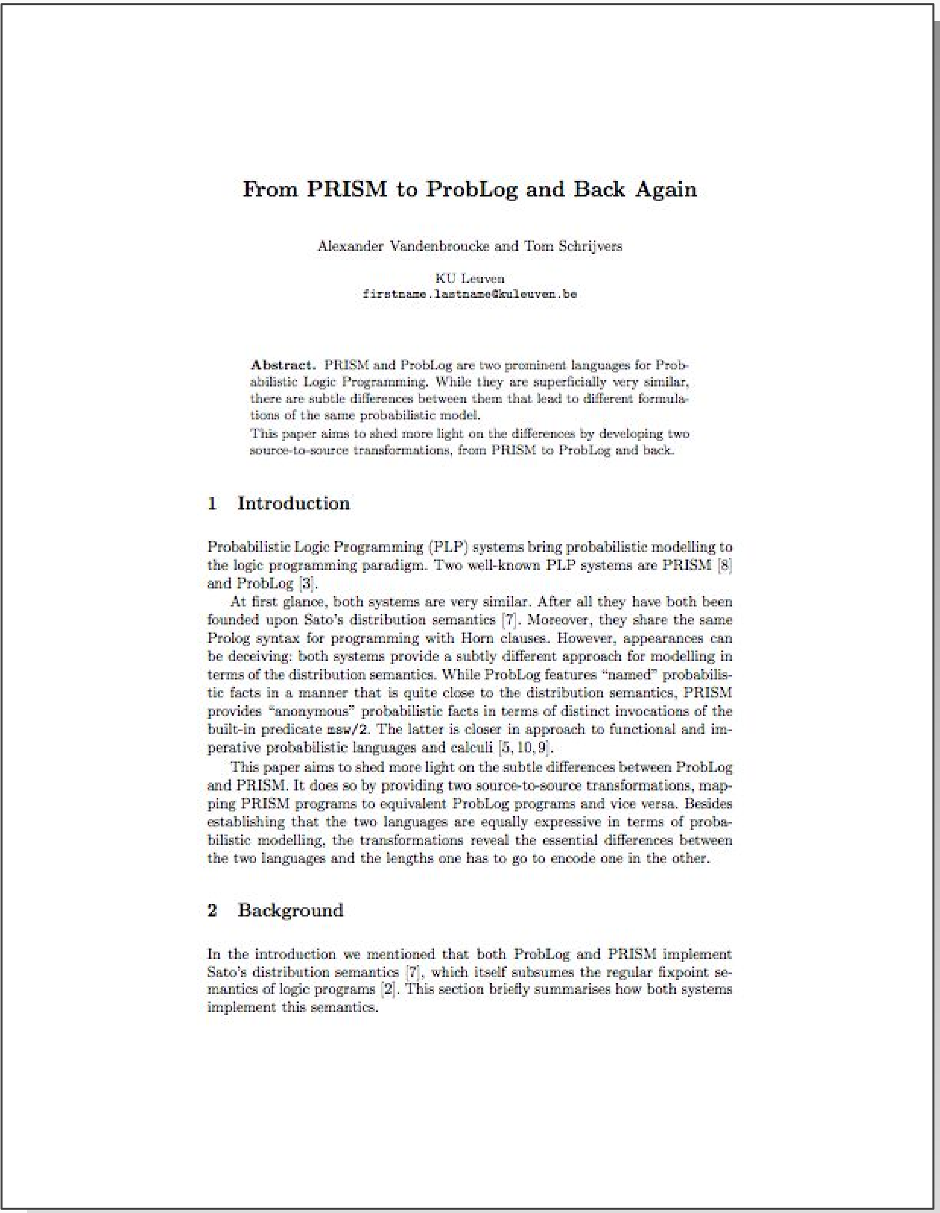
Future Work - Performance

PRISM evaluation is highly efficient

↕

ProbLog is #P **worst-case**

Simulating ProbLog in PRISM could add an exponential factor **worst-case**



From PRISM to ProbLog and Back Again

Alexander Vandenbroucke and Tom Schrijvers

KU Leuven
firstname.lastname@kuleuven.be

Abstract. PRISM and ProbLog are two prominent languages for Probabilistic Logic Programming. While they are superficially very similar, there are subtle differences between them that lead to different formulations of the same probabilistic model.
This paper aims to shed more light on the differences by developing two source-to-source transformations, from PRISM to ProbLog and back.

1 Introduction

Probabilistic Logic Programming (PLP) systems bring probabilistic modelling to the logic programming paradigm. Two well-known PLP systems are PRISM [8] and ProbLog [3].

At first glance, both systems are very similar. After all they have both been founded upon Sato's distribution semantics [7]. Moreover, they share the same Prolog syntax for programming with Horn clauses. However, appearances can be deceiving: both systems provide a subtly different approach for modelling in terms of the distribution semantics. While ProbLog features "named" probabilistic facts in a manner that is quite close to the distribution semantics, PRISM provides "anonymous" probabilistic facts in terms of distinct invocations of the built-in predicate msw/2. The latter is closer in approach to functional and imperative probabilistic languages and calculi [5, 10, 9].

This paper aims to shed more light on the subtle differences between ProbLog and PRISM. It does so by providing two source-to-source transformations, mapping PRISM programs to equivalent ProbLog programs and vice versa. Besides establishing that the two languages are equally expressive in terms of probabilistic modelling, the transformations reveal the essential differences between the two languages and the lengths one has to go to encode one in the other.

2 Background

In the introduction we mentioned that both ProbLog and PRISM implement Sato's distribution semantics [7], which itself subsumes the regular fixpoint semantics of logic programs [2]. This section briefly summarises how both systems implement this semantics.

Read The Paper:

⟶ Implementation details

⟶ More examples

⟶ Background

Questions