On the Relationship between PRISM and CLP($\mathcal{BN}$)

Vítor Santos Costa VSC@DCC.FC.UP.PT
CRACS and DCC-FCUP, Universidade do Porto, Portugal

Aline Paes AMPAES@COS.UFRJ.BR
Department of Systems Eng. and Computer Science, Universidade Federal do Rio de Janeiro, Brazil

1. Introduction

The last few years have seen significant progress in Statistical relational learning (SRL) (Getoor & Taskar, 2007) and/or Probabilistic Inductive Logic Programming (De Raedt et al., 2008). This progress requires the design and implementation of languages that can be used to express uncertainty over relational data, motivating the development of a very large number of languages in this area. These languages differ on both the representation they use for structure, with logic-based formalisms being quite popular, and on the representations for uncertainty, with both directed and undirected graphical models being used.

Recently, there has been interest on understanding the relationship between these different languages. We observe that establishing a mapping between two different languages often provides insight in understanding the languages themselves. Moreover, such a mapping may be useful in sharing implementation technology and experience. For example, PRISM has been used to implement SLPs (Muggleton, 1995).

One important category of SRL languages are the languages based in logic programming. Very different examples include PHA (Poole, 1993), SLPs (Muggleton, 1995), CLP($\mathcal{BN}$) (Santos Costa et al., 2003; Santos Costa et al., 2008), and ProbLog (De Raedt et al., 2007). PRISM (Sato & Kameya, 2001) is one particularly important example that provides a nice, compact, formalism for describing directed graphical models. Thus, PRISM can benefit both from the progress in inference on bayesian networks, and from progress in the implementation of sequential logic programming languages, namely B-Prolog.

In this work, we study the relation between CLP($\mathcal{BN}$) and PRISM. Although CLP($\mathcal{BN}$) is less widely used than PRISM, it also benefits from an implementation. The two languages are clearly quite related, and in fact our original motivation was to investigate whether there was a simple mapping between the two languages. We will focus on the PRISM to CLP($\mathcal{BN}$) mapping first. Our approach is example-based: we will start from standard PRISM examples, and try to see whether they can be *"elegantly"* mapped to CLP($\mathcal{BN}$).

2. Background

We quickly introduce PRISM and CLP($\mathcal{BN}$) next.

2.1. PRISM

PRISM (PRogramming in Statistical Modelling) is a logic-based modelling language, originally proposed by Sato (Sato & Kameya, 2001), and based on *distribution semantics*. In these models, probabilities are given to set of ground atoms. In PRISM programming, these atoms are embodied through the $\texttt{msw}(i, n, v)$ which can be enunciated as "switch named i randomly taking the value v at trial n".

PRISM is a natural language for describing stochastic processes. It can also be used in a nice way to describe HMMs and stochastic grammars, with a sophisticated battery of algorithms having been developed to address both inference and learning (Sato & Kameya, 2001).

It is important to notice that in practice, one often drops trial n and assumes every *call* to the switch will be independent. In this case, the semantics of PRISM become very dependent of *what is a call.* PRISM relies on tabling execution, and we will often need to assume a *single variant call to each goal.*

2.2. CLP($\mathcal{BN}$)

CLP($\mathcal{BN}$) (Constraint Logic Programming for Bayesian Networks) is a language to model graphical models, and namely bayesian models. CLP($\mathcal{BN}$) programs are set of Prolog clauses, where some clauses may contain $\mathcal{BN}$ constraints. $\mathcal{BN}$ constraints are of the form {`X = Key with PT`}, where X must be a Prolog variable, Key a term, and PT a probability distribution. The answer substitution to a CLP($\mathcal{BN}$) query will therefore contain a set of constraints: together these constraints will form a bayesian network defining a probability distribution over the possible answers to the query. Notice that every variable X with the same key *must assume the same value.*

Evidence is a major concern in CLP($\mathcal{BN}$). Evidence may be introduced in a this formalism by unifying a constrained variable with a value. Probabilities are then computed over the joint network.

3. Static Bayesian Networks

The connection between CLP($\mathcal{BN}$) and PRISM is very clear in static bayesian networks. Consider PRISM's implementation of the *alarm* example:

```
world(Fi,Ta,Al,Sm,Le,Re) :-
   msw(fi,Fi),          % P(Fire)
   msw(ta,Ta),          % P(Tampering)
   msw(sm(Fi),Sm),      % CPT P(Smoke | Fire)
   msw(al(Fi,Ta),Al),   % CPT P(Alarm | Fire,Tampering)
   msw(le(Al),Le),      % CPT P(Leaving | Alarm)
   msw(re(Le),Re).      % CPT P(Report | Leaving)
```

This program defines a joint distribution over Fi, Ta, Al, Sm, Le, Re. From the CLP($\mathcal{BN}$) point of the view, this simply corresponds to a network of constraints. Thus, the program is a CLP($\mathcal{BN}$) program if one defines `msw` as follows:

```
msw(S,V) :-
   term_variables(S, Vs),
   parameters(S, Domain, Matrix),
   { V = S with p(Domain,Matrix,Vs) }.
```

The first subgoal finds the ancestors for the current random variable V. The second subgoal uses the `values` and `set_sw` declarations to construct the conditional distributions. Last, we constrain V.

This nice result suggests that PRISM could mapped in a straightforward fashion to CLP($\mathcal{BN}$). In fact, one needs not to construct very sophisticated models to obtain major differences between the two languages.

4. Stochastic Processes

The `dcoin` program is a PRISM program that simulates a stochastic process where we can throw one of two coins:

```
dcoin(N,Coin,[R|Rs]) :-
   N > 0,
   msw(Coin,R),
   ( R == head, NextCoin = coin(2)
   ; R == tail, NextCoin = coin(1) ),
   N1 is N-1,
   dcoin(N1,NextCoin,Rs).
dcoin(0,_,[]).
```

notice that the two coins will have different distributions. To understand this program as a CLP($\mathcal{BN}$) program, we need to ask *what are the random variables.* PRISM provides a very compact encoding.

First, we clearly have N different $Coin$ variables. Notice that the previous mapping would not construct a single variable: this is because each call to `msw` is independent, and thus a different random variable. This principle may be enforced by generating new keys at every new call:

```
msw(S,V) :-
   term_variables(S, Vs),
   parameters(S, Domain, Matrix),
   new_trial(S,K),
   { V = K with p(Domain,Matrix,Vs) }.
```

where `new_trial` generates a new key, related to S.

This is not the main difference, though. If we observe carefully, R is conditioned on a random variable, and is therefore a random variable itself (although with a deterministic conditional probability table). The correct CLP($\mathcal{BN}$) program is therefore:

```
dcoin(N,Coin,[R|Rs]) :-
   N > 0,
   msw(Coin,R),
   msw(R,NextCoin),
   N1 is N-1,
   dcoin(N1,NextCoin,Rs).
dcoin(0,_,[]).
```

notice that the program is deceptively simple. In order to know the domain and the CPTs, `msw` must be able to distinguish between $Coin$ and R. This requires consulting the current constraint store (the details are available in the YAP CLP(BN) distribution).

5. Hidden Markov Models

Notice that PRISM can use values of random variables to switch. This is clearly a problem for CLP($\mathcal{BN}$), as demonstrated when trying to understand PRISM's very nice profile Hidden Markov Model (pHMM) example. pHMMs are a popular approach to represent conserved subquences in families of proteins, they can be coded as:

```
hmm([],end).
hmm(Sequence,State) :-
   State \== end,
   msw(move_from(State),NextState),
   msw(emit_at(State), Symbol),
   ( Symbol = epsilon ->
       hmm( Sequence, NextState )
   ; Sequence = [Symbol|TailSeq],
       hmm( TailSeq , NextState )
   ).
```

Sequence are strings of letters, where letters may be bases or amino-acids. Two states are special: *start* and *end*. Other states can be of the form $m(I), d(I), i(I), 1 < I < L$ where L is the number of conserved positions. An $m(I)$ state corresponds to a match, and may transition to $i(I), m(I+1), d(I+1)$. $i(I)$ and $m(I)$ states emit a normal symbol, $d(I)$ symbols do not.

This program is naturally understood as specifying a path in the unrolled markov model. Each call to a `msw` corresponds to a branch in the path: either moving from a state to the next states, or whether emitting specific symbols. A more subtle poin is that the program *requires tabling*: if we reach the same state with the same sequence from two different callers, we should obtain a *single* random variable, not two independent trials.

To translate the program to CLP($\mathcal{BN}$) we need the following transformations:

1. Random Variables are not about state, but about *transitions*. To encode this in CLP($\mathcal{BN}$) we need to pass both the current state (a non-random variable) and the current transition (a random variable).

2. Given that the state s not a random-variable, we can declare whether a symbol will be emitted or not without much ado.

3. Last, CLP($\mathcal{BN}$) would construct the full HMM. This requires transforming backtracking on recursion.

We cannot present the transformed program in the abstract, due to lack of space. The program will work correctly without tabling, but will take exponential time. Tabling `hmm/3` obtains dynamic programming, as usual, and scales up to sequences with hundreds on the example with $L = 17$.

6. Conclusions

We compare CLP($\mathcal{BN}$) and PRISM on a number of standard PRISM examples. The examples show that PRISM can provide a very compact encoding of interesting statistical models. Simpler examples can be adapted to CLP($\mathcal{BN}$) easily. More complex and dynamic examples require understanding how the random variables affect the possible states in the model explicitely.

Arguably, the main advantage of PRISM encodings is that they are very compact and concise. Arguably, the main disadvantage of PRISM encodings is that they are very compact and concise. In our experience, PRISM does provide very short and elegant descriptions, but that may not be always straightforward to understand, especially as when needs to understand the interplay between switches, independence, and tabled execution. CLP($\mathcal{BN}$) makes it harder to connect control and random variables; on the other hand, what is a random variable and what is not is clearer.

Automatic translation from the core fragment of CLP($\mathcal{BN}$) to PRISM seems easier than the other way round. On the other hand, CLP($\mathcal{BN}$) has a very different way to condition on static evidence that may be hard to code with PRISM.

Acknowledgments This work has been partially supported by projects JEDI (PTDC/EIA/66924/2006) and STAMPA (PTDC/EIA/67738/2006) and funds granted to CRACS through the Programa de Financiamento Plurianual, FCT and Programa POSC. Aline Paes would like to thank the Brazilian Council for Scientific and Technological Development (CNPq) for the financial support.

References

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming — theory and applications*, vol. 4911 of *Lecture Notes in Artificial Intelligence*. Springer.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). ProbLog: A probabilistic Prolog and its application in link discovery. *IJCAI* (pp. 2462–2467).

Getoor, L., & Taskar, B. (Eds.). (2007). *Statistical relational learning*. The MIT press.

Muggleton, S. (1995). Stochastic logic programs. *ILP*.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, *64*, 81–129.

Santos Costa, V., Page, C. D., & Cussens, J. (2008). *Probabilistic inductive logic programming*, chapter CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge, 156–188. Springer-Verlag.

Santos Costa, V., Page, D., Qazi, M., & Cussens, J. (2003). CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge. *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)* (pp. 517–524). Acapulco, Mexico.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res. (JAIR)*, *15*, 391–454.