Modeling Dependent Events with CHRiSM for Probabilistic Abduction

Henning Christiansen[1] and Amr Hany Saleh[2]

[1] Research group PLIS: Programming, Logic and Intelligent Systems
Department of Communication, Business and Information Technologies
Roskilde University, P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: henning@ruc.dk
[2] Department of Computer Science and Engineering
Faculty of Media Engineering and Technology
The German University in Cairo, Egypt
E-mail: amr.shehata@student.guc.edu.eg

Abstract. Most earlier approaches to probabilistic abductive logic programming are based on the assumption that abducibles represent independent events. This enables efficient and incremental calculation of probabilities, but may not be suitable for all real world problems. As an attempt to introduce such dependencies in a logic programming setting, we have applied CHRiSM, which is a recent probabilistic extension to Constraint Handling Rules, for specification and evaluation of probability distributions over dependent abducibles. It is shown that this principle integrates well with earlier work on probabilistic abduction based on CHR, generalizing it to handle such dependencies. We present our first experiments with a working implementation that show the potential for interesting applications. On the other hand, our experience underlines problems concerning high complexity due to lacking incremental methods for probability calculation.

1 Introduction

Abduction refers here to the process of reasoning to find explanations (unknown facts), that can explain given observations, and often this definition is refined so we search for a *best* explanation; see, e.g., [8] for more background. Abduction has been studied in the context of logic programming, and adding probabilities provides a way of giving priorities to different explanations, thus providing a meaningful characterization of a "best" solution. With few exceptions, earlier work on probabilistic abduction in logic programming has been based on the assumption that abducible facts are instances of mutually independent random variables. This implies a restricted expressive power but also a straightforward way of calculation probabilities in an incremental way. Earlier work [3] in this direction has shown that the language of Constraint Handling Rules, CHR [9], is well suited for describing implementations with a variety of characteristics, e.g., adapting for best-first search. Here we apply a recent probabilistic extension CHRiSM [21] to extend this work with dependent probabilities, using a

To appear in *Proceedings of the 8th International Workshop on Constraint Handling Rules*
September 8, 2011, Cairo, Egypt.

specific form of CHRiSM programs as plug-in modules to characterize such joint probability distributions over abducibles and explanation.

The approach handles negation of abducibles and points forward to a treatment of non-ground and existentially quantified abducibles, which often cause problems in implementations of abductive logic programming. We can also point to other open problems: the calculation of probabilities does not scale well to large sets of abducibles, and useful operations such as subsumption and entailment checking seem to require a detailed knowledge about the actual probability distribution over possible worlds which may not be easily captured. Also, while our semantics in a natural way captures programs with potentially infinitely many abducibles, the way we use CHRiSM excludes this option.

In section 2, we introduce a probabilistic possible worlds semantics as a general theoretical setting for probabilistic logic programming and use it to define abstract syntax and semantics for such programming languages. In section 3, we introduce CHRiSM [21] and describe our approach for using it to define the desired probability distributions. Section 4 explains how probability distributions defined using CHRiSM can be integrated with CHR programs that implement search over the clauses of the given probabilistic logic program, including best-first. Section 5 discusses a few possible enhancements of these strategies. Section 6 discusses selected related work and section 7 gives a conclusion, including suggestions for future work.

2 Basic Concepts

We have chosen a probabilistic possible worlds semantics as a firm theoretical basis, which makes it possible for us to derive – rather than postulate – properties that are reasonable for probability distributions used for the abducibles in a probabilistic abductive logic program. This semantics introduced below is similar to the one used by [16] and adapted to here to our specific case.

2.1 Probabilistic Possible Worlds

We consider here a world (or state of affairs) as given by the truth values of a set of world properties. Intuitively, this set may be so huge that it is impossible for an observer to perceive or name them all in any feasible way. For technical simplicity, we assume the set of world properties to be countable, but larger cardinalities could have been allowed at the price of replacing the summations in the following definition by integrals or other limit constructions. There are no assumptions about probabilistic independency of such properties.

Definition 1. *Assume a set of* world properties $\mathcal{X}$. *A* world *is a mapping from each* $x \in \mathcal{X}$ *to* $\{true, false\}$*; a world* w *may also be viewed as a set of literals, called* world literals*, containing* x *when* x *is true in* w *and* $\neg x$ *otherwise.*

A probability distribution *over worlds is a mapping* P *into* $[0; 1]$ *with*

$$\sum_{w \in W} P(w) = 1.$$

A world w with $P(w) > 0$ is called a possible *world. The set of possible worlds is denoted $\mathcal{W}$. A* sub-world *s is a finite set of world literals with, for no $x \in \mathcal{X}$, both $x \in s$ and $\neg x \in s$; any property x with neither $x \in s$ or $\neg x \in s$ is* undefined *in s. For sub-world s and possible world w, we write $w \models s$ iff $x \in w$ for any $x \in s$ and $x \notin w$ for any $\neg x \in s$; the* world set *for s is defined as $\mathcal{W}(s) = \{w \in \mathcal{W} \mid w \models s\}$.*

The probability of a sub-world s is defined as follows.

$$P(s) = \sum_{w \in \mathcal{W}(s)} P(w)$$

A sub-world s with $P(s) > 0$ is called a possible *sub-world.*

The notion of a sub-world may model what is relevant or perceivable for an observer under the given circumstances, which means that is not interesting or perhaps not possible for the observer to distinguish between different elements of $\mathcal{W}(s)$ for a given sub-world s. Notice that the probability of a sub-world is given by a typically infinite but countable sum which is well-defined due to the initial assumptions.

Whenever a world literal y is the negation $\neg x$ of world property x, we let $\neg y$ refer to x, and the usage of y being undefined in some s means the same as x being undefined in s. It the rest of this section, we assume that a set of world properties $\mathcal{X}$ with a probability distribution P is given. We observe the following properties that follow immediately from the definition.

Proposition 1. *Let s_1, s_2 and s be sub-worlds, x a world literal undefined in s, and let $\emptyset$ refer to the empty sub-world.*

- *Whenever $s_1 \subseteq s_2$, we have that $\mathcal{W}(s_2) \subseteq \mathcal{W}(s_1)$.*
- *Whenever $s_1 \subseteq s_2$, we have that $P(s_1) \geq P(s_2)$.*
- *Whenever $s_1 \subseteq s_2$ and s_1 is not possible, neither is s_2.*
- $P(s) = P(s \cup \{x\}) + P(s \cup \{\neg x\})$
- $P(\emptyset) = 1$.
- *There exists at least one possible world.*

The following property states that a suitable, joint probability distribution over a set of logical literals is compatible with a probabilistic possible world semantics.

Proposition 2. *Let M be a set of ground atoms induced by given sets of predicates and function symbols, and let S be the collection of all sets of non-contradictory literals made from a subset of M. Let, furthermore, P be a function $S \to [0; 1]$ that satisfies the following conditions.*

- *Whenever, for $s_1, s_2 \in S$ with $s_1 \subseteq s_2$, it holds that $P(s_1) \geq P(s_2)$.*
- *Whenever, for $s \in S$ and x being a literal of M with $x, \neg x \notin S$, it holds that $P(s) = P(s \cup \{x\}) + P(s \cup \{\neg x\})$.*
- $P(\emptyset) = 1$.

Then M and P' induces a probabilistic possible world semantics, cf. def. 1, where P' is the restriction of P to 2^M. Furthermore, the generalization of P' to sub-worlds coincides with P.

2.2 Probabilistic Abductive Logic Programs

Here we define the abstract syntax and semantics of probabilistic abductive logic programs, ProbALPs, and characterize some of their important properties. No explicit integrity constraints are included as they can be embedded in the probability distribution by setting zero probability for undesired combinations of abducibles.

Disjoint and countably infinite sets of constants, function symbols, variables and predicates are assumed as usual, together with two disjoint sets of predicates, *abducibles* and *defined* (the latter a.k.a. *program* or *user defined*). Terms and atoms are built in the usual way; a *program literal* is either a defined atom or a perhaps negated abducible atom. Instances and ground instances are defined in the usual way. We assume a *naming function* that maps any ground abducible atom into a unique world property.

Definition 2. *Given a probabilistic possible world semantics, cf. def. 1, and the nomenclature introduced above, a* probabilistic abductive logic program, ProbALP, *is a finite set of* program clauses*, each of the form*

$$h \mathbin{:-} b_1, \ldots, b_n$$

where h is a defined atom called the head*, $n \geq 0$ and $b_1, \ldots, b_n$, called the* body*, consisting of program literals. A* query *is a conjunction of ground atoms. An* explanation *is a finite existentially quantified set of abducible literals; the quantifier is as usual left implicit in our notation.*

A standard completion semantics [4] is assumed for program clauses. Thus, for a ProbALP *Prog* and explanation E, it is well-defined, whether a given query is true in *Prog* extended with E, denoted $Prog, \exists E \models A$.

A ground explanation is, via the naming function, considered equivalent to a sub-world, and the notion of world sets is generalized accordingly. The world set $\mathcal{W}(E)$ for an explanation E is defined as the union of the world sets for each ground instance of E, and the probability $P(E)$ is defined as $\sum_{w \in \mathcal{W}(E)} P(w)$.

For any two explanations E_1 and E_2 with $\mathcal{W}(E_2) \subseteq \mathcal{W}(E_1)$, we say that E_1 *subsumes* E_2 and that E_1 is more general than E_2. An explanation E is *consistent* whenever $P(E) > 0$ (or, equivalently, that $\mathcal{W}(E) \neq \emptyset$). Whenever $\mathcal{W}(E_1) = \mathcal{W}(E_2)$, we say that E_1 and E_2 are *equivalent*. Whenever E_1 subsumes E_2 and they are not equivalent, we say that the subsumption is *strict*.

The following example indicates that subsumption in the context of dependent abducibles is more complicated than in the independent case.

Example 1. Independently of the underlying set of possible worlds, we have that $\{a\}$ subsumes $\{a, b\}$, and $\{a(X)\}$ subsumes $\{a(17)\}$, both judgments made by purely syntactic arguments. Whether $\{a(X), b(17)\}$ subsumes $\{c(12)\}$ (or the reverse for that matter) depends on the specific semantics, and in general we cannot expect a decision procedure for such judgments.

We observe the following properties about subsumption that follow immediately from the definition.

Proposition 3. *Whenever E_1 subsumes (resp. strictly subsumes, and is equivalent with) E_2, it holds that $P(E_1) \geq P(E_2)$ (resp. $P(E_1) > P(E_2)$, and $P(E_1) = P(E_2)$).*

Whenever E is an explanation and A is an abducible literal, it holds that E is equivalent with $E \cup \{A\}$ iff $P(E) = P(E \cup \{A\})$.

We do not employ these properties in our implementation, but they demonstrate that an effective procedure for evaluation probabilities may give rise to a way of simplifying the presentation of explanations into some minimal form.

Definition 3. *Consider a given ProbALP Prog and a query Q. An* abductive answer for *(or* explanation of*) Q is a consistent explanation E such that for any possible world $w \in \mathcal{W}(E)$ it holds that*

$$Prog, w \models Q.$$

An explanation E of Q is minimal *if not subsumed by another explanation of Q.*

Notice that we used a semantic characterization for minimality rather than a syntactic one based on, e.g., the number of literals or a subset relationship.

Abductive procedures, such as those we describe in this paper, tend to produce minimal proof explanations. In order to avoid a lengthy, technical definition, we introduce this notion informally: a *proof explanation* for a given query Q is an explanation E which is generated through a specific proof of Q (using an SLD proof rule) by incorporating into E those abducible literals that are encountered in the clause instances that are applied.

Example 2. Consider a program with abducible predicates a and b consisting of the two clauses $\{p\texttt{:-}a, b;\ p\texttt{:-}a, \neg b\}$. Then $\{a\}$ is a minimal explanation of p, and $\{a, b\}$ as well as $\{a, \neg b\}$ are minimal proof explanations.

In the case of independent abducibles, proof minimal explanations may be compacted into minimal ones by syntactic rules based on opposite literals as inherent in this example, cf. [3]. However, for dependent abducibles, it requires obviously additional rules that depend on the underlying possible world semantics.

3 Using CHRiSM to Define Possible Worlds Semantics

3.1 An Overview of CHRiSM

CHRiSM [21] is an extension to CHR with probabilistic rules and a probabilistic semantics based on the probabilistic-logic language PRISM [19].

Derivation is seen as a probabilistic process, in the sense that different rules have a certain probability of firing. In this way each derivation is assigned a probability, and the probability of a particular final state is calculated from all possible derivation leading to it. In these calculations, all choices made by CHRiSM are considered independent, but as we see below, a CHRiSM program can define a variety of dependent distributions over sets of constraints.

CHRiSM adds two types of probability statements; it gives the possibility of adding a probability of firing a rule as well as probabilities for selection among the alternatives of a disjunction in the body of the rule. For example:

```
0.7 ?? a ==> c.
a ==> 0.8 ?? b ; c.
```

The first rule indicates a probability of 0.7 to fire in a state that contains the constraint `a`. The second one indicates a probability of 0.8 to add constraint `b` (and not `c`) and probability 0.2 to add constraint `c` (and not `b`) to a state that contains the constraint `a`. CHRiSM provides two sorts of queries for probabilities.

1. `prob( (`Q `<==>` A`) , P)`. Assigns to variable `P` the probability that a derivation starting from an initial state Q ends in a final state whose constraints are exactly those given by A.
2. `prob( (`Q `===>` A`) , P)`. Assigns to `P` the probability that a derivation starting from an initial state Q ends in a final state that contains as a subset those given by A.

In addition, A may contain negated constraints $\sim C$, with the meaning that the constraint C should not appear in the final states considered; this principle is called negation by absence. For more details about CHRiSM, see [21].

3.2 Probability Distributions for ProbALPs using CHRiSM

Our approach to use CHRiSM to define probabilities is to write a CHRiSM program of a specific form that we call a *Probability Defining CHRiSM program*, PDCP. Such a program includes the following constraints,

- a constraint for each abducible predicate with the same arity,
- a special constraint `start/0`,
- a special constraint `failure/0` that must not appear in any rule.

To fit with current limitations in the CHRiSM system, a program should be range-restricted[3] so that no non-ground constraints are created during its execution. Intuitively `start` serves as a "world creator" that defines all possible final states of interest in a probabilistic way, whereas `failure` is never included in such a final state, meaning that $\sim$`failure` via negation as absence characterizes all non-failed, final states. An explanation should be represented as a conjunction of literals without duplicates, and with the negative ones indicated by CHRiSM's negation as absence.

Provided that 1) abducibles always are ground, and 2) the CHRiSM program never produces a failure or loops, we can take the following CHRiSM query as a definition of $P(E)$.

(i) `prob((start ===>` E`), P)`

[3] A program is range-restricted whenever any variable in a rule body appears also in the head of that rule.

I.e., the variable `P` is bound to $P(E)$. We observe that the function P fulfills the conditions of proposition 2, so the probability distribution thus defined is compatible with a possible world semantics. To see this, notice that CHRiSM sums probabilities over exactly those final states that play the role of $\mathcal{W}(E)$.

Example 3. Consider a the following CHRiSM program viewed as PDCP.

```
0.5 ?? start ==> a.
0.1 ?? a ==> b.
```

It indicates a dependency between `a` and `b`: logically it embeds $\mathtt{b} \rightarrow \mathtt{a}$ (i.e., if you have `b`, you must also have `a`). It gives, for example, $P(\{\mathtt{a}\}) = 0.5$ and $P(\{\mathtt{b}\}) = 0.05$, whereas $P(\{\mathtt{a}, \mathtt{b}\}) = 0.05$. From this, we can see the following.

- Abducibles `a` and `b` are indeed dependent as $P(\{\mathtt{a}, \mathtt{b}\}) \neq P(\{\mathtt{a}\}) \times P(\{\mathtt{b}\})$.
- Proposition 3 confirms that explanations $\{\mathtt{a}, \mathtt{b}\}$ and $\{\mathtt{b}\}$ are equivalent.

This can be generalized to PDCPs that apply failure for integrity checking. This involves the problem that some probability mass is lost, i.e., the sum of probabilities for final and non-failed states is < 1. However, when we normalize the probability found by `prob` using the probability of reaching a non-failed final state, the conditions of definition 2 are established. More precisely, the following query will bind variable `P` to the desired probability of explanation E.

(ii) `prob((start ===> ~failure), PN), prob((start ===>` E`), P1), P is P1/PN.`

The following example illustrates the use of failure for integrity checking as well as more elaborate CHRiSM rules than those of the previous example.

Example 4. We consider a PDCP that may be used for reasoning about the weather in a period of three days covering `yesterday`, `today` and `tomorrow`. The weather can be either `sunny` or `rainy` a particular day, but not both (in other word, the classification describes the prevailing weather that day). The `weather/2` constraint represents abducibles relating weather and days.

```
start ==> 0.6 ?? weather(sunny,yesterday) ; weather(rainy,yesterday).
start ==> 0.6 ?? weather(sunny,today)     ; weather(rainy,today).
start ==> 0.6 ?? weather(sunny,tomorrow)  ; weather(rainy,tomorrow).

0.7 ?? weather(X,yesterday), weather(X,today) ==> weather(X,tomorrow).
weather(X,D), weather(Y,D) ==> X=Y.
```

The first three CHRiSM rules state a particular "background probability" for each weather type, 0.6 for sunny and $1 - 0.6 = 0.4$ for rainy. The fourth rule increases the probability for a given weather if it has lasted for two days already, and the last rule is a CHR rule indicating that only one type of weather is possible on a particular day.

It can bee seen that some derivations beginning from `start` lead to failure. If, for example, the first three rules produce `weather(rain,yesterday)`,

`weather(rain,today)` and `weather(sunny,tomorrow)`, and the fourth rule decides to fire, the derivation fails and we loose the probability mass $0.4 \times 0.4 \times 0.6 \times 0.7$. It can be verified that the following query, which is the part of *(ii)* that calculates the normalization factor,

```
prob((start ===> ~failure), PN)
```

produces the value $1-(0.4\times0.4\times0.6\times0.7+0.6\times0.6\times0.4\times0.7) = 0.832$. Using the probability distribution defined by *(ii)* and the PCDP above, we obtain, for example, $P($`weather(sunny,yesterday)`$) = P($`weather(sunny,today)`$) = 0.4$ and $P($`weather(sunny,tomorrow)`$) \approx 0.6404$. The increase in the overall probability for sun tomorrow is an effect of the the fourth rule that emphases the general majority for sunny weather.

As the factor used for normalization is specific for a given PDCP, we need only calculate it once for that program. The following example shows the code lines needed for an implementation, which also fixes a problem with the currently available implementation of CHRiSM.

Example 5. In the current implementation of CHRiSM, version 0.2 (downloaded August 2011), the `prob` predicate fails when there are no final states for the given query and where it, according to the specification of [21] referred above, should succeed with probability 0.

The following predicate `probNorma` corrects for this problem and incorporates the normalization needed for programs with failures, cf. query *(ii)* above. The purpose of the `init` predicate is to have the normalization factor evaluated only once, prior to the use of `probNorma`.

```
probNorma(Cs,PN):-
    (prob((start ===> Cs),P) -> true ; P=0),
    non_failure_prob(N),
    PN is P/N.

:- dynamic non_failure_prob/1.

init:-
    (prob((start ===> ~failure),P) -> true ; P=0),
    asserta(non_failure_prob(P)).
```

3.3 Experiments using CHRiSM for Non-ground Explanation

The currently available implementation of CHRiSM is only guaranteed to provide correct results for ground queries, and the precise meanings of queries *(i)* and *(ii)* above for non-ground E are not obvious from [21]. We made some promising experiments with the available CHRiSM implementation, which involved small change in one of CHRiSM's internal utilities.

Queries with variables in positive literals do sum the probabilities of the correct set of final states, including when sharing occurs. For example:

```
prob((start ===> a(X),b(X)), N)
```

Here exactly those final states are counted in which there is a pair of abducibles `a(`x`),b(`x`)` for some term x.

Negative, non-ground literals are handled correctly when preceded by a positive literal containing the same variable. For example:

```
prob((start ===> a(X), ∼b(X)), N)
```

Here exactly those final states are counted in which there is an abducibles `a(`x`)` and no `b(`x`)` for the same term x.

For the ground case that we have studied above, we emphasized that duplicate constraints must be avoided in explanations when we give them to the `prob` predicate. This is due to fact that tests for membership in final states made by `prob` is designed according to the multiset semantics of CHRiSM. So for example `prob((start===>a,a),P)` will only count states that include two or more occurrences of `a`. This problem appears also when we have an explanation of the form $E_1 = \{\texttt{a(X)}, \texttt{a(1)}\}$. According to our semantics, this explanation is equivalent with $E_2 = \{\texttt{a(1)}\}$, and to get the correct probability from `prob`, we must use the form E_2 for reasons we just pointed out. However, we should not remove such internally subsumed literals in our interpreters for ProbALPs: it may be the case that an explanation such as E_1 is specialized by `X` being instantiated to, say, `2` so the explanation becomes $\{\texttt{a(2)}, \texttt{a(1)}\}$ which is not subsumed by E_2 which is unaffected by the instantiation of `X`.

4 Integration of Probability Distributions in CHRiSM into Interpreters for ProbALP in CHR

Implementations in CHR of probabilistic abduction with independent probabilities for the abducibles have been described in [3]. Abductive logic programs are compiled into query interpreters in CHR that produce proof minimal explanations. One of the advantages of using CHR is a flexible control which makes it easy to adopt these query interpreters to work best first, calculating the most probable answers in order of decreasing probability, as long as the user asks for more solutions.

We sketch here the methods of [3], adapting them to an example of a simplistic ProbALP; only the calculation of probabilities differs. The referenced paper gives correctness proofs for its special case, which seems straightforward to adapt to the present setting. Here we proceed in an informal way.

We consider an example program with defined predicates `p/1`, `q/1`, ..., and abducibles `a/1`, ... It is assumed that the abducibles and their joint probability distribution is defined in terms of PDCP as described in section 3. The logic program includes among others the following clauses for `p/1`.

```
p(X):- q(X), a(X).
p(1).
```

We start explaining a query interpreter that works depth-first and which requires the ProbALP to be range-restricted, i.e., abducibles and other predicate calls are consistently ground.

The fundamental constraint in the interpreter is the following `explain/3`. An instantiated constraint in the store represents a branch of a computation that proceeds in an adapted SLD manner. The meaning of a call of the form

`explain(`Q`, `E`, `P`)`

is that subquery Q remains in order to have found a proof for the initial query; E is the partial explanation of abducibles encountered so far, and P is the probability of E. Both Q and E are given as lists of literals. When Q has become empty, E is a proof minimal explanation for the initial query. Before giving the details, we define a suitable top-level predicate and the CHR rule that applies when an explanation has been found.

```
explain(Q):-  explain(Q, [], 1).

explain([],E,P) <=>
     print explanation E with its probability P.
```

The program clauses for predicate `p/1` is compiled into the following CHR clause.

```
explain( [p(X)|G], E, P) <=>
    rename([p(X)|G]+E,[p(Xr1)|Gr1]+E1),
    explain([q(Xr1),a(Xr1)|Gr1], E1, P),
    (X=1 -> explain(G, E, P) ; true).
```

This rule applies for a branch with a query having `p(...)` as its "current" subgoal and a continuation referred to by the variable `G`. It rewrites the given branch into new branches, potentially one for each clause in the definition of `p/1`. The renaming predicate `rename(`A`,`A'`)` assigns to A' a copy of A whose variables are replaced in a consistent manner by new variables, and should be called for each such new branch in order to avoid cluttering up alternative instantiations of the variables. Notice, however, that for the last one, we can bypass renaming as the involved variables anyhow are not used for other purposes.

As shown in the example rule above, a clause with a distinct variable in each of its arguments (here: only one) is translated into a straightforward rewriting of the query. The last clause, being that fact `p(1)` implies a unification which may fail, and if that happens, the branch will silently disappear.

Executing an abducible predicate means to add it to the current explanation (if it is not there already) and re-calculate the probability. An abducible predicate is compiled into a CHR rule as follows plus a completely analogous one for negated abducibles.

```
explain([a(X)|G],E,P) <=>
    (member(a(X),E) -> E1 = E ; E1 = [a(X)|E]),
    prob_ex(E1, P1),
    explain(G,E1,P1).
```

The `prob_ex` predicate is an auxiliary predicated that does a little formatting and calls the CHRiSM program to obtain the probability of the new explanation. Notice that it removes duplicate literals before continuing. The `probNorma` predicate is the auxiliary shown in example 5 above which generates a suitable call to CHRiSM's `prob`.

```
prob_ex(E, P):-
    turn the list E into a conjunction EConj using commas,
    probNorma(EConj, P).
```

What has been shown so far is sufficient to turn any range-restricted ProbALP into a depth-first query interpreter. In order to produce a best-first interpreter, we can keep the same basic structure but add a few additional auxiliary constraints to ensure that only the `explain` constraint with the highest probability (in its third argument) can be involved in a rule application; this presents no conceptual difficulties, and we refer to [3] for the details.

Example 6. We consider the PDCP introduced in example 4 and extend it with two constraints `weekday` and `tomorrow`, and the following rule.

```
start ==> 0.7143 ?? workday(tomorrow) ; weekend(tomorrow).
```

It defines a probability of `tomorrow` being either a workday or a day in the weekend which, however, cannot both be the case at the same time. Together with the following clauses it forms a ProbALP.

```
plan_for_tomorrow(work):-  weather(rainy,today), weather(rainy,tomorrow).

plan_for_tomorrow(beach):- weekend(tomorrow), weather(sunny,tomorrow).

plan_for_tomorrow(beach):- weather(sunny,yesterday),
                           weather(sunny,today), weather(sunny,tomorrow).
```

We have compiled this into a best-first query interpreter along the lines above, and here we show some queries and answers. For the following query, one answer is produced.

```
?- explain([plan_for_tomorrow(work)])
Most probably solution:
  [weather(rainy,tomorrow),weather(rainy,today)]
  P=0.192307692307692
Another and less probable explanation? ;
No (more) solutions
```

The next query leads to two different solutions, one for each of the relevant program clauses.

```
?- explain([plan_for_tomorrow(beach)]).
Most probably solution:
  [weather(sunny,tomorrow),weather(sunny,today),weather(sunny,yesterday)]
  P=0.259615384615385
Another and less probable explanation? ;
```

```
Most probably solution:
  [weather(sunny,tomorrow),weekend(tomorrow)]
  P=0.182957884615385
Another and less probable explanation? ;
No (more) solutions
```

It should be noticed that these two solutions overlap in the sense that they have a possible world in common, as their union is consistent. Finally, if we extend the query as follows with an "observed" abducible, we miss out the first of these of these answers as there is no final state containing both `workday(tomorrow)` and `weekend(tomorrow)`.

```
?- explain([plan_for_tomorrow(beach),workday(tomorrow)]).
Most probably solution:
  [workday(tomorrow),weather(sunny,tomorrow),weather(sunny,today),
       weather(sunny,yesterday)]
  P=0.185443269230769
Another and less probable explanation? ;
No (more) solutions
```

This probability is smaller than for the analogous solution for the previous query as the contribution for the world containing `weekend(tomorrow)` plus `weather(sunny, ···)` for all three days.

5 Enhancements for Defining Probabilities using CHRiSM

In this section we show useful extensions to PDCPs that are easy to integrate with the implementation principle shown above. It is shown how a notion of domains may be associated with specific arguments of abducible predicates in order to ensure range-restrictedness. We introduce also an observation module that allows to introduce pre-observed abducibles in the initial constraint store when probabilities are computed.

5.1 Domains that Ensure Ground Abducibles

Consider an example concerning a set of girls $\{Ann, Sue, Eva, \ldots\}$ of size n. A PDCP should be used to describe the probability of each girl being blonde. In principle this may be done using n rule, one for each girl. To avoid this, we might naively suggest the following rule which is not range-restricted, and whose logical meaning does not either express the intended meaning.

```
0.7 ?? start ==> blonde(X).
```

To cope with such unbound variables, we can introduce domain declarations and associate a domain to specific arguments exemplified as follows.

```
domain(girls, [ann, sue, eva, tia]).
type(blonde, 1, girls).
```

The last declaration associates the domain `girls` to the first argument of the abducible predicate `blonde`. With these declarations, we can automatically compile the knowledge of the domains into to relevant PDCP rules, exemplified as follows.

```
0.7 ?? start, constant(girls,X) ==> blonde(X).
```

To get everything to work, the domain declarations should be incorporated into the probability queries performed by the query interpreters as follows.

```
prob((constant(girls,ann),constant(girls,eva),...,start ===> E),P).
```

This saves typing and results in a better program structure, but does not scale well with respect to efficiency for large domains.

5.2 Pre-Observed Facts Module

By pre-observed facts we refer to abducibles that are known to hold before a query is given to some ProbALP. These may be utilized in our implementation by including them in the initial states for probability queries. Assuming, for example, that abducibles `a` and `b` are pre-observed, the probability queries should be done as follows.

```
prob((a, b, start ===> E), P).
```

This may in some cases reduce complexity by removing early in the computation all states that are inconsistent with `a` and `b`. For some PDCPs, the probabilities found in this way and normalized accordingly corresponds to conditional probabilities, but in general we cannot count on this.

6 Comparison with Related Work

Abductive logic programming has been studied since the early 1990ies; see [6, 11] for an overview. Extending with probabilities as a means to prioritize among alternative explanations has been treated by several authors in varying settings, mostly under the assumption of independent probabilities for abducibles. Among the most influential work, we find Poole's which stems also from the early 1990ies [15]. Our own recent work on expressing probabilistic abduction in CHR (with independent probabilities) extends previous work with integrity constraints and the option of integrating with external constraint solvers; see [3] which also gives a detailed overview of related work concerned with abduction using CHR. PRISM [19] is a highly generic probabilistic-logic language which can work with an unlimited number of independent random variables and as a special case it can also express abduction (with independent abducibles). There are other recent works on abduction in different logic-based frameworks that are not directly comparable to ours, and which allow probabilistic dependencies [17, 2].

The notion of Possible Worlds has been used in philosophy and linguistics in attempts to cover pragmatic aspects of meaning, i.e., the relationship between representation and the real world or collections of hypothetical worlds (or states of affairs); see, e.g., the work by Stalnaker as summarized in [22]. It is used by [13] to explain the semantics of counterfactual statements and is standard for modal logics [12]. Probabilistic possible worlds have been proposed by several authors. It is difficult to trace a single origin, but we may mention some influential papers, e.g., [14, 1, 7]. The formulation we have used is similar to [16] that provides a concise introduction and overview; see also [10]. Although our terminology is different, there is also a strong resemblance to Sato's distribution semantics [18] and an earlier work by Dantsin [5].

Several of the referenced and other similar works state to have a possible worlds semantics but quite often this is not formalized or described in detail. In the present paper, we used possible worlds as the defining framework to characterize desired properties for the probability distributions, and our distributions using CHRiSM was shown to conform with those.

Recent work by Simari and Subrahmanian [20] describes an approach to a special form of abduction in a limited logic language with intervals of probabilities associated to clauses. It is based on a sort of possible worlds, each being a collection of derived properties (which is unconventional), and it stresses also the absence of independency assumptions. However, the setting is quite different from ours and most other definitions of Abductive Logic Programming, and a direct comparison is difficult.

We are not aware of other work than our own that uses a probabilistic possible world semantics to define dependent distributions over abducible predicates, which are applied for generation of probability weighted explanations for given observations. In addition, we propose a specific method for defining such probability distributions.

7 Conclusion

In this work, we have defined a semantics of probabilistic abductive logic programs that supports dependent probabilities for abducibles. We introduced a special kind of CHRiSM programs called *probability defining CHRiSM programs*, which is consistent with this semantics. There is an appealing similarity between the way a possible world semantics sums up probabilities over possible worlds and the way CHRiSM sums probabilities over final states. However, this way of using CHRiSM is not very efficient and blocks effectively for using an unlimited number of abducibles.

Our theoretical model can handle both negated and non-ground abducibles, but the current version of CHRiSM cannot handle our encoding of non-ground abducibles, except in a few simple cases. We may anticipate that future releases of CHRiSM provide a clarified semantics in these respects.

We have also shown how such probability defining CHRiSM programs can be incorporated into query interpreters written in CHR. Moreover, we consid-

ered facilitating the definition of probability distributions using CHRiSM with the introduction of domains for abducible predicates and suggested a way to incorporate pre-observed abducible facts.

7.1 Time Complexity Problems

Our implementation of probability calculations uses CHRiSM in a way that forces the enumeration of all possible final states each time a probability is asked for. This introduces a factor into the overall time complexity which is exponential in the number of abducibles.

There may be several ways to approach this problem. In principle we could generate all possible worlds (i.e, final CHRiSM states) once and for all before the program execution starts, and represent them as an array of bit vectors and provide a fast matching to determine in which worlds a given (partial) explanation is actually true; this may be a viable solution in specific cases which even may be supported by specialized hardware. However, the development of incremental methods may be more suited as explanations gradually get specialized by adding more literals or unifying variables; this is how it is typically done for independent probabilities, but for the dependent case, there is no obvious universal solution. The notion of unambiguous CHRiSM programs suggested by [21] may be a useful restriction in the search for incremental methods.

7.2 Operations on Explanations

Subsumption and entailment of explanations as well as compaction of different explanations into more general (and thus more probable) ones can be computed efficiently for independent abducibles, as e.g., studied by [3]. However, our use of CHRiSM as a sort of blackbox makes this difficult, since these operations depend on properties embedded in the possible worlds semantics. Clearly more research needs to be done here.

Acknowledgements. This work is supported by the project "Logic-statistic modelling and analysis of biological sequence data" funded by the NABIIT program under the Danish Strategic Research Council.

References

1. John C. Bigelow. Possible worlds foundations for probability. *Journal of Philosophical Logic*, 5(3):299–320, 1976.
2. Jianzhong Chen, Stephen Muggleton, and José Carlos Almeida Santos. Learning probabilistic logic models from probabilistic examples. *Machine Learning*, 73(1):55–85, 2008.
3. Henning Christiansen. Implementing probabilistic abductive logic programming with Constraint Handling Rules. In Tom Schrijvers and Thom W. Frühwirth, editors, *Constraint Handling Rules*, volume 5388 of *Lecture Notes in Computer Science*, pages 85–118. Springer, 2008.

4. Keith L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Perseus Publishing, 1978.
5. Evgeny Dantsin. Probabilistic logic programs and their semantics. In Andrei Voronkov, editor, *RCLP*, volume 592 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 1991.
6. Marc Denecker and Antonis C. Kakas. Abduction in logic programming. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, pages 402–436. Springer, 2002.
7. Ronald Fagin and Joseph Y. Halpern. Uncertainty, belief, and probability. *Computational Intelligence*, 7:160–173, 1991.
8. Peter A. Flach and Antonis C. Kakas, editors. *Abduction and Induction: Essays on their relation and integration.* Kluwer Academic Publishers, April 2000.
9. Thom Frühwirth. *Constraint Handling Rules.* Cambridge University Press, August 2009.
10. Joseph Y. Halpern. A logical approach to reasoning about uncertainty: a tutorial. In Xabier Arrazola, Kepa Korta, and Francis Jeffry Pelletier, editors, *Discourse, Interaction, and Communication*, pages 141–155. Kluwer, 1998.
11. A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of Logic in Artificial Intelligence and Logic Programming,* vol. 5, Gabbay, D.M, Hogger, C.J., Robinson, J.A., (eds.), Oxford University Press, pages 235–324, 1998.
12. Saul Kripke. Semantical considerations on modal logics. *Acta Philosophica Fennica*, 16:83–94, 1963.
13. David Lewis. *Counterfactuals.* Blackwells, 1973.
14. Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.
15. David Poole. Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Computing*, 11(3):377–400, 1993.
16. David Poole, Alan Mackworth, and Randy Goebel. *Computational intelligence: a logical approach.* Oxford University Press, 1998.
17. Sindhu Raghavan and Raymond Mooney. Bayesian abductive logic programs. In *Proceedings of the AAAI-10 Workshop on Statistical Relational AI (Star-AI 10)*, pages 82–87, Atlanta, GA, July 2010.
18. Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In Leon Sterling, editor, *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming*, pages 715–729. Leon Sterling, 1995.
19. Taisuke Sato. A glimpse of symbolic-statistical modeling by PRISM. *J. Intell. Inf. Syst.*, 31(2):161–176, 2008.
20. Gerardo I. Simari and V. S. Subrahmanian. Abductive inference in probabilistic logic programs. In Manuel V. Hermenegildo and Torsten Schaub, editors, *ICLP (Technical Communications)*, volume 7 of *LIPIcs*, pages 192–201. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.
21. Jon Sneyers, Wannes Meert, Joost Vennekens, Yoshitaka Kameya, and Taisuke Sato. CHR(PRISM)-based probabilistic logic learning. *Theory and Practice of Logic Programming*, 10(4-6):433–447, 2010.
22. Robert Stalnaker. On the representation of context. *Journal of Logic, Language and Information*, 7(1):3–19, 1998.