Probabilistic Legal Reasoning in CHRiSM

JON SNEYERS, DANNY DE SCHREYE
Dept. of Computer Science, KU Leuven, Belgium
(*e-mail:* `FirstName.LastName@cs.kuleuven.be`)

THOM FRÜHWIRTH
University of Ulm, Germany
(*e-mail:* `thom.fruehwirth@uni-ulm.de`)

submitted February 2012; revised X; accepted X

Abstract

Riveret et al. have proposed a framework for probabilistic legal reasoning. Their goal is to determine the chance of winning a court case, given the probabilities of the judge accepting certain claimed facts and legal rules.

In this paper we tackle the same problem by defining and implementing a new formalism, called probabilistic argumentation logic, which can be seen as a probabilistic generalization of Nute's defeasible logic. Not only does this provide an automation of the — only hand-performed — computations in Riveret et al, it also provides a solution to one of their open problems: a method to determine the initial probabilities from a given body of precedents.

1 Introduction

Riveret, Rotolo et al. (2007) have proposed a framework of probabilistic legal reasoning based on the argumentation framework of Prakken and Sartor (1997), which provides a dialectical proof theory in the formal setting of Dung (1995). Their goal is to determine the chance of winning a court case, given the probabilities of the judge accepting certain claimed facts to be valid and legal rules to be applicable.

Roth, Riveret et al. (2007) tackle a similar problem, but with the focus on finding legal strategies for the involved parties to maximize their chances of winning. They propose a rather complex framework, based on a logic layer, an argument layer, a dialectical layer, a procedural layer, and finally a probabilistic weighting.

To our knowledge, these approaches have not been implemented so far. Both papers only contain hand-performed computation on an example as a proof of concept. The implementation of an 'argument assistance system' is left explicitly as future work.

Another issue is how to determine or verify the probabilities, which are assumed to be known in advance. Riveret, Rotolo et al. (2007) suggest that maybe in future work, some kind of statistical analysis of the judge's past decisions could be performed.

In this paper we tackle the problem by defining a new formalism, called probabilistic argumentation logic, which can be seen as a probabilistic generalization of defeasible logic. Our formalism is inspired by Nute (2001)'s defeasible logic, but instead of using a precedence (priority) relation to settle conflicts between contradictory defeasible rules, we use explicit assumptions for this purpose. In some sense, our proposal can also be seen as a form of abductive reasoning (Kakas, Kowalski et al. 1992).

We formalize our approach and implement it in CHRiSM (Sneyers, Meert et al. 2010). CHRiSM is a rule-based probabilistic logic programming language based on Constraint Handling Rules (Frühwirth 2009) in the host language PRISM (Sato 2008).

Our implementation provides an automation of the probability computations that were hand-performed in (Riveret, Rotolo et al. 2007). The built-in learning algorithm of CHRiSM also provides a solution to the open problem of determining the initial probabilities from a given body of precedents.

The remainder of this paper is organized as follows. Section 2 briefly introduces CHRiSM. Then, in Section 3 we introduce the approach of Riveret, Rotolo et al. (2007) and the running example in their paper. We show how this example can be encoded in CHRiSM. Section 4 introduces our new formalism, probabilistic argumentation logic. We implement it in CHRiSM. In Section 5 we demonstrate how the CHRiSM program can be used for learning. Finally we conclude in Section 6.

2 CHRiSM

Constraint Handling Rules (Frühwirth 2009; Sneyers, Van Weert et al. 2010; Frühwirth and Raiser 2011) is a high-level language extension based on multi-headed rules. Being a language *extension*, CHR is implemented on top of an existing programming language, which is called the *host language*. An implementation of CHR in host language X is called CHR(X). Several CHR(Prolog) systems are available.

PRISM (PRogramming In Statistical Modeling) is a probabilistic extension of Prolog (Sato 2008). It supports several probabilistic inference tasks, including sampling, probability computation, and expectation-maximization (EM) learning.

In (Sneyers, Meert et al. 2010), a new rule-based probabilistic logic programming language was introduced, called CHRiSM — short for **CH**ance **R**ules **i**nduce **S**tatistical **M**odels. It is based on CHR(PRISM) and it combines the advantages of CHR and those of PRISM. Like CHR, it is a very concise and expressive programming language. Like PRISM, it has built-in support for several probabilistic inference tasks. Furthermore, CHRiSM rules can be freely mixed with CHR rules and Prolog clauses.

Syntax and Informal Semantics. A CHRiSM program $\mathcal{P}$ consists of a sequence of *chance rules*. Chance rules rewrite a multiset $\mathbb{S}$ of data elements, which are called (CHRiSM) *constraints* (mostly for historical reasons). Syntactically, a constraint `c(`X_1`,..,`X_n`)` looks like a Prolog atom: it has a functor `c` of some arity n and arguments X_1`,..,`X_n which are Prolog terms. The multiset $\mathbb{S}$ of constraints is called the *constraint store* or just *store*. The initial store is called the *query* or *goal*, the final store (obtained by exhaustive rule application) is called the *answer* or *result*.

A chance rule is of the form: "`P ?? Hk \ Hr <=> G | B.`", where `P` is a probability expression, `Hk` is a conjunction of (kept head) constraints, `Hr` is a conjunction of (removed head) constraints, `G` is a guard condition (a Prolog goal to be satisfied), and `B` is the body of the rule. If `Hk` is empty, the rule is called a *simplification* rule and the backslash is omitted; if `Hr` is empty, the rule is called a *propagation* rule and it is written as "`P ?? Hk ==> G | B`". The guard `G` is optional; if it is removed, the "`|`" is also removed. The body `B` is a conjunction of CHRiSM constraints, Prolog goals, and probabilistic disjunctions (defined in (Sneyers, Meert et al. 2010); we do not need them here).

Intuitively, the meaning of a chance rule is as follows: If the constraint store $\mathbb{S}$ contains elements that match with the head of the rule, and furthermore, the guard `G` is satisfied, then we can consider rule application. The subset of $\mathbb{S}$ that corresponds to the head of the rule is called a rule *instance*. Depending on a coin flip with a probability given by `P`, the rule instance is either ignored or it actually leads to a rule application. Every rule instance may only be considered once. A rule with probability 1 corresponds to a regular CHR rule; the "`1 ??`" may be dropped.

Rule application has the following effects: the constraints matching `Hr` are removed from the constraint store, and then the body `B` is executed, that is, Prolog goals are called and CHRiSM constraints are added into the store.

Operational Semantics. The abstract operational semantics $\omega_t^{??}$ of a CHRiSM program $\mathcal{P}$ is given by a state-transition system that resembles[1] the abstract operational semantics ω_t of CHR (Sneyers, Van Weert et al. 2010). We refer to (Sneyers, Meert et al. 2010) for the formal definition of $\omega_t^{??}$. Just like CHR, CHRiSM can also be given a refined operational semantics, where conjunctions are evaluated depth-first from left to right, considering occurrences of the "active" constraint from top to bottom.

Observations. A full observation `Q <==> A` denotes that there exist a series of probabilistic choices such that a derivation starting with query `Q` results in the answer `A`. A partial observation `Q ===> A` denotes that some answer for query `Q` contains at least `A`: in other words, `Q ===> A` holds iff `Q <==> B` with `A` $\Subset$ `B`.

The following PRISM built-ins can be used to query a CHRiSM program:

- `sample Q` : execute the query `Q` while making probabilistic choices;
- `prob Q <==> A` : compute the probability that `Q <==> A` holds, i.e. the chance that the choices are such that query `Q` results in answer `A`;
- `prob Q ===> A` : compute the probability that an answer for `Q` *contains* `A`;
- `learn(L)` : do expectation-maximization learning from observations `L`

3 The mad cow example and dialogue games

The running example in (Riveret, Rotolo et al. 2007) is the following. John, the proponent, wants to sue Henry, the opponent, claiming compensation (c) for the damage that Henry's cow caused to him when he drove off the road to avoid the cow. John argues that an animal's owner has to pay damages caused by their animal, that (a) Henry is the owner of the cow, and that (b) the accident was caused by the need to avoid the cow (argument A). Henry can counterattack in various ways: he can claim that (d) the damage was due to John's negligence (he did not pay sufficient attention to crossing animals) – argument B – or that (e) it was a case of force majeure: the cow suddenly went crazy and crossed into the street – argument C. The last objection could be replied to by using the debated rule that only exogenous events count as force majeure, and (f) the cow's madness is endogenous (argument D).

[1] If all rule probabilities are 1 — i.e. if the CHRiSM program is actually just a regular CHR program — then the $\omega_t^{??}$ semantics boils down to the ω_t semantics of CHR.

Riveret, Rotolo et al. (2007) assume an abstract argumentation framework (Dung 1995), which consists of a set of arguments and a binary "defeats" relation. They then define the notion of a dialogue game which captures the rules of legal argumentation. A dialogue is a sequence of abstract arguments, in which the proponent and the opponent alternate, each argument defeats the previous argument, and the proponent cannot repeat arguments and has to strictly defeat the preceeding argument of the opponent.

Moreover, each of the arguments has some given "construction chance", which is the probability that the judge will actually accept the argument. The aim is to estimate the overall chance that the case is won.

CHRiSM encoding of the example. For reasons of space, we omit the CHRiSM encoding of dialogue games with abstract arguments, which are treated as black boxes that only interact through the "defeats" relation. It can be found in a companion paper (Sneyers, De Schreye et al. 2013). Here we immediately proceed with a finer level of granularity.

The arguments in (Riveret, Rotolo et al. 2007) consist of premises and rules, which each have a probability of being accepted by the judge. For example, argument A consists of the premises that Henry owns the cow (a), and that the accident was caused by the need to avoid the cow (b), together with the rule "$a \wedge b \rightarrow c$", where c stands for "Henry has to compensate damages". If a will certainly be accepted, b is accepted with a probability of 0.9, and the rule "$a \wedge b \rightarrow c$" is certainly accepted, then the overall construction chance of argument A is $1 \times 0.9 \times 1 = 0.9$.

We will call both premises and conclusions "statements" and use ground Prolog terms to denote them. The auxiliary predicate `neg`/2 simply negates a literal in a way that avoids double negations.

We use a dummy predicate `begin`/0 (to be used as the initial goal). The main constraint predicate is `accept`/2, which indicates that the judge conditionally accepts some statement: `accept(S,C)` denotes that statement `S` is accepted if all conditions `C` (a Prolog list of statements) hold. If `C` is the empty list, the statement is unconditionally accepted; otherwise the acceptance of the statement can still be retracted if one of the conditions turn out to be unacceptable. If a statement is already accepted with conditions `A`, then it is redundant to also accept it with stronger conditions `B` $\supseteq$ `A`.

```
accept(X,A) \ accept(X,B) <=> subset(A,B) | true.
```

The above rule implies a set semantics for unconditionally accepted statements.

A condition is redundant if it is implied by the other conditions:

```
accept(A,B) \ accept(X,C) <=> select(A,C,D), subset(B,D) | accept(X,D).
```

If a statement `Y` was accepted with conditions `B`, but one of those conditions is contradicted ("undercut") by a statement `X` with weaker conditions `A` that are implied by `B`, then the acceptance of `Y` has to be retracted — we use a simpagation rule to remove the `accept(Y,B)` constraint:

```
accept(X,A) \ accept(Y,B) <=>
        subset(A,B), neg(X,NX), member(NX,B) | true.
```

Finally, we allow no contradictions:

```
accept(X,A), accept(Y,B) ==> subset(A,B), neg(X,Y) | fail.
```

Now we encode the premises and the rules of the arguments:

```
% Argument A (rule r1, premises a and b):
% "If Henry is the owner of the cow (a) and the accident was caused by the
% need to avoid the cow (b), then Henry has to compensate damages (c)."

1.0 ?? accept(a,[]), accept(b,[]) ==> accept(c, [app(r1)]).  % r1
1.0 ?? begin ==> accept(a,[]).
0.9 ?? begin ==> accept(b,[]).
```

The rule being used is a defeasible rule, so its conclusion `c` will be accepted with the condition that the rule is actually applicable (`app(r1)`). This condition can be "undercut" by arguments B or C.

```
% Argument B (rule r2, premise d):
% "If John was negligent (d), then r1 is not applicable."
0.8 ?? accept(d,[]) ==> accept(not(app(r1)),[]).             % r2
0.5 ?? begin ==> accept(d,[]).
```

For example, the judge could accept `a`, `b` and `d` and both rules, to reach the state "`accept(c,[app(r1)]), accept(not(app(r1)),[])`". Now the undercutting rule removes the conditional acceptance of `c`, because its condition was contradicted.

```
% Argument C (rule r3, premise e):
% "If the cow was mad (e), it was a case of force majeure.
% so r1 is not applicable."
0.5 ?? accept(e,[]) ==> accept(not(app(r1)), [app(r3)]).     % r3
0.2 ?? begin ==> accept(e,[]).
```

Again, the above rule (`r3`) is defeasible, so its conclusion can only be accepted with the condition `app(r3)`, which can be undercut by the final argument:

```
% Argument D (rule r4, premise f):
% "If the cow's madness is endogenous (f), then the 'force majeure'
% rule r3 is not applicable."
0.5 ?? accept(f,[]) ==> accept(not(app(r3)), []).            % r4
0.3 ?? begin ==> accept(f,[]).
```

The only thing now left to do, is to resolve the conditions by making assumptions. For example, one possible result of the query `begin` is the following:

```
accept(a, []), accept(b, []), accept(e, []),
accept(c, [app(r1)]), accept(not(app(r1)), [app(r3)])
```

In this case, both `c` and `not(app(r1))` are conditionally accepted. Although `not(app(r1))` undercuts the conditions of `c`, the "undercut" rule is not applicable (yet) because the condition `[app(r3)]` is not weaker than the condition `[app(r1)]`. However, since there is no counter-evidence for `app(r3)`, we can assume this condition to hold, "promoting" `not(app(r1))` to an unconditionally accepted statement, which then causes the acceptance of `c` to be retracted.
To implement this idea, we add the following rules at the end of the program:

```
begin <=> assume.
assume, accept(X,C) ==> select(A,C,D), accept(A,[]) ; true.
assume <=> true.
```

The middle rule nondeterministically selects conditions and accepts them. This can either succeed or lead to contradiction (i.e. there is counter-evidence), causing backtracking.

This concludes the program. We can compute the desired probability – which took several pages of manual calculations in (Riveret, Rotolo et al. 2007) – with a simple query:

```
?- prob begin ===> accept(c,[]).
Probability of begin ===> accept(c,[]) is: 0.494100000000000
```

4 Generalization and Formalization

In order to generalize the running example, we will propose a transformation from an arbitrary probabilistic legal argumentation logic $\mathcal{A}$ — a notion that will be introduced in this section — to a CHRiSM program $P_{CHRiSM}(\mathcal{A})$.

4.1 Probabilistic Argumentation Logic

We use $lit(A)$ to denote the set of literals over a set of atomic formulas A, i.e. $lit(A) = A \cup \{\neg a \mid a \in A\}$. We will sometimes denote conjunctions over literals as sets since the order of the conjuncts is irrelevant.

Definition 4.1 (*Probabilistic argumentation logic*)
A *probabilistic argumentation logic* or PAL is a tuple (S, A, R, P), where S is a set of statements and A is a set of assumptions (with $S \cap A = \emptyset$), R is a set of rules, and P is a function assigning probabilities to each rule, $P : R \mapsto [0, 1]$. The rules in R have the following form:

$$s_1 \wedge \ldots \wedge s_n \Rightarrow c_1 \wedge \ldots \wedge c_m \text{ assuming } a_1 \wedge \ldots \wedge a_k$$

where the left hand side (the *antecedent*) is a conjunction of literals ($s_i \in lit(S \cup A)$) which can be empty ($n \geq 0$), the right hand side (the *consequent*) is a non-empty ($m \geq 1$) conjunction of literals ($c_i \in lit(S \cup A)$, and the part after "assuming" (the *assumption*) is a possibly empty ($k \geq 0$) conjunction of assumption literals ($a_i \in lit(A)$). If the left hand side is empty, the rule is also called a *fact* and the arrow can be omitted; if the part after "assuming" is empty, the rule is called *unconditional* and the keyword "assuming" can be omitted.

To illustrate the definition, we now write out the running mad cow example as a formal probabilistic argumentation logic

$$\mathcal{A}_{\text{mc}} := (\{a, b, c, d, e, f\}, \{\texttt{app(r1)}, \texttt{app(r3)}\}, R_{\text{mc}}, P_{\text{mc}})$$

where the rules $R_{\text{mc}} = \{r_1, r_2, r_3, r_4, s_a, s_b, s_d, s_e, s_f\}$ are the following:

$$\begin{array}{rrcl} & r_1 & := & a \wedge b \Rightarrow c \text{ assuming } \texttt{app(r1)} \\ & r_2 & := & d \Rightarrow \neg\texttt{app(r1)} \\ & r_3 & := & e \Rightarrow \neg\texttt{app(r1)} \text{ assuming } \texttt{app(r3)} \\ & r_4 & := & f \Rightarrow \neg\texttt{app(r3)} \\ \forall x \in \{a, b, d, e, f\} : & s_x & := & x \end{array}$$

and the probabilities P_{mc} are the following:

$$P_{\mathrm{mc}} := \{(r_1, 1), (r_2, 0.8), (r_3, 0.5), (r_4, 0.5), (s_a, 1), (s_b, 0.9), (s_d, 0.5), (s_e, 0.2), (s_f, 0.3)\}$$

Now we define an interpretation of a PAL as a set of conditional statements.

Definition 4.2 (*Conditional statement*)
Given a PAL $\mathcal{A} = (S, A, R, P)$, a conditional statement is a pair (s, C) with $s \in \mathit{lit}(S \cup A)$ and $C \subseteq \mathit{lit}(A)$, such that C is not self-contradictory, that is, there is no $c \in A$ such that both $c \in C$ and $\neg c \in C$.

Definition 4.3 (*Interpretation*)
Given a PAL $\mathcal{A}$, an interpretation is a set I of conditional statements of $\mathcal{A}$ such that if $(s, C_1) \in I$ and $(\neg s, C_2) \in I$, then $C_1 \not\subseteq C_2$ and $C_2 \not\subseteq C_1$.

In other words, the conditional statements are not directly contradictory — although both a statement and its negation can be conditionally accepted at the same time, as long as the assumptions are different.

Definition 4.4 (*Partial Ordering of Interpretations*)
We say an interpretation I_1 is smaller than an interpretation I_2, denoted $I_1 \leq_i I_2$, if for all conditional statements $(s, c_1) \in I_1$, there is a corresponding conditional statement $(s, c_2) \in I_2$ such that $c_1 \subseteq c_2$.

Note that $I_1 \leq_i I_2$ implies that the set of statements in I_1 is a subset of the statements in I_2, so I_1 makes less claims than I_2 (which is why we call it smaller), but the claims in I_1 are in a sense stronger since they have a weaker condition.

We now define the semantics of a PAL, somewhat inspired by the definitions in (Nute 2001), but extending them to take the explicit conditions into account, as well as the rule selection (which will be needed to introduce the rule probabilities).

Definition 4.5 (*Compliant Interpretation w.r.t. Rule Selection*)
Given a PAL $\mathcal{A} = (S, A, R, P)$ and a set of selected rules $R_s \subseteq R$, a compliant interpretation I w.r.t. the selected rules R_s is a minimal (w.r.t. $\leq_i$) interpretation that satisfies the following additional criterion:

- if R_s contains a rule $r = (S_r \Rightarrow C_r$ assuming $A_r)$,
- and $\forall s \in S_r : \exists c : (s, c) \in I$ (the antecedent is conditionally accepted),
- and $\forall a \in A_r : \neg\exists c : (\neg a, c) \in I$ (the assumption is not questioned),
- then for every set $\{(s_1, c_1), \ldots, (s_n, c_n)\} \subseteq I$ such that $S_r = s_1 \wedge \ldots \wedge s_n$, it must hold that $\forall x \in C_r : \exists y \subseteq A_r \cup \{c_1, \ldots, c_n\} : (x, y) \in I$ (the consequent is conditionally accepted).

Definition 4.6 (*Valid Interpretation w.r.t. Rule Selection*)
Given a PAL $\mathcal{A} = (S, A, R, P)$ and a set of selected rules $R_s \subseteq R$, a valid interpretation I w.r.t. the selected rules R_s is a compliant interpretation without gratuitous statements, that is, there is no subset $K \subseteq I$ with $K \neq \emptyset$ such that:

- if R_s contains a rule $r = (S_r \Rightarrow C_r$ assuming $A_r)$,
- and $\forall s \in S_r : \exists c : (s, c) \in I \setminus K$, and $\forall a \in A_r : \neg\exists c : (\neg a, c) \in I$,

- then for every set $\{(s_1, c_1), \ldots, (s_n, c_n)\} \subseteq I \setminus K$ such that $S_r = s_1 \wedge \ldots \wedge s_n$, it must hold that $\forall x \in C_r : \forall y \subseteq A_r \cup \{c_1, \ldots, c_n\} : (x, y) \notin K$.

Returning to the mad cow example, consider the rule selection R_{mc} of all rules. The following is the only valid interpretation w.r.t. R_{mc}:

$$\{(a,\emptyset),(b,\emptyset),(d,\emptyset),(e,\emptyset),(f,\emptyset),(\neg\texttt{app(r1)},\emptyset),(\neg\texttt{app(r3)},\emptyset)\}$$

This is the only valid interpretation w.r.t. $R' = \{r_1, r_3, r_4, s_a, s_b, s_e, s_f\}$:

$$\{(a,\emptyset),(b,\emptyset),(c,\emptyset),(e,\emptyset),(f,\emptyset),(\neg\texttt{app(r3)},\emptyset)\}$$

An interpretation like the above, but with $(c, \{\texttt{app(r1)}\})$ instead of $(c, \emptyset)$ also satisfies the criterion of Def. 4.5, but it is not compliant because it is not minimal w.r.t. $\leq_i$. The condition for the acceptance of the consequent of applicable rules is allowed to be weaker than (i.e. a subset of) the union of all the conditions arising from the antecedent and assumption. This relaxation serves two goals. Firstly, it means that if a consequent can be derived in different ways such that it would be accepted multiple times with varying conditions, it suffices to have only the weakest conditions in the interpretation. Secondly, because the interpretation has to be minimal w.r.t. $\leq_i$, conditions will only be present in the interpretation to avoid contradiction. The following example illustrates this point.

r_1 := `bird` $\Rightarrow$ `flies` assuming `normal-bird`
r_2 := `tux` $\Rightarrow$ `bird` $\wedge$ `tuxedo-plumage` assuming `eyes-OK`
r_3 := `tuxedo-plumage` $\Rightarrow$ `penguin` assuming `feathers-make-bird`
r_4 := `penguin` $\Rightarrow \neg$`flies`
r_5 := `tux`

Consider the selection of all rules. The interpretation $\{(\texttt{tux}, \emptyset),\ (\neg\texttt{eyes-OK}, \emptyset)\}$ is compliant, but it is not a valid interpretation since the statement $\neg$`eyes-OK` is trivially gratuitous: there is not even a rule that could derive it.

The following interpretation satisfies the criterion of Def. 4.5:

$$\left\{\begin{array}{c} (\texttt{tux},\emptyset),\quad (\texttt{bird},\{\texttt{eyes-OK}\}),\quad (\texttt{flies},\{\texttt{eyes-OK},\texttt{normal-bird}\}),\\ (\texttt{tuxedo-plumage},\{\texttt{eyes-OK}\}),\quad (\texttt{penguin},\{\texttt{eyes-OK},\texttt{feathers-make-bird}\}),\\ (\neg\texttt{flies},\{\texttt{eyes-OK},\texttt{feathers-make-bird}\}) \end{array}\right\}$$

but it is not minimal w.r.t. $\leq_i$; we can relax some conditions to get a minimal interpretation, for example: (in this case there are three minimal interpretations)

$$\left\{\begin{array}{c} (\texttt{tux},\emptyset),\quad (\texttt{bird},\emptyset),\quad (\texttt{flies},\{\texttt{normal-bird}\}),\\ (\texttt{tuxedo-plumage},\emptyset),\quad (\texttt{penguin},\{\texttt{eyes-OK}\}),\quad (\neg\texttt{flies},\{\texttt{eyes-OK}\}) \end{array}\right\}$$

Note that some conditions have to be kept in order to avoid a direct contradiction between `flies` and $\neg$`flies`. Also note that in the above program, one could replace r_4 with `penguin` $\Rightarrow \neg$`normal-bird`, $\neg$`flies` to get rid of these conditions and have a unique minimal interpretation which contains $(\neg\texttt{flies}, \emptyset)$.

Definition 4.7 (*Plausibility of a statement*)
Given a PAL $\mathcal{A} = (S, A, R, P)$, a statement literal $s \in \mathit{lit}(S)$ is called *plausible* w.r.t. a rule selection R_s if all valid interpretations w.r.t. R_s contain a conditional statement (s, c) for some c.

Definition 4.8 (*Acceptability of a statement*)
Given a PAL $\mathcal{A} = (S, A, R, P)$, a statement literal $s \in \mathit{lit}(S)$ is called *acceptable* w.r.t. a rule selection R_s if it is plausible and there exists a valid interpretation w.r.t. R_s which contains the conditional statement $(s, \emptyset)$.

In the above example, `tux`, `bird`, and `tuxedo-plumage` are acceptable statements, while `flies`, ¬`flies`, and `penguin` are plausible but not acceptable. All other literals are not even plausible.

The probability $\mathit{prob}(R_s)$ of a rule selection $R_s \subseteq R$ is defined as follows:

$$\mathit{prob}(R_s) = \left(\prod_{r \in R_s} P(r) \right) \left(\prod_{r \in R \setminus R_s} 1 - P(r) \right)$$

which ensures that $\sum_{R_s \in \mathcal{P}(R)} \mathit{prob}(R_s) = 1$.

Definition 4.9 (*Probability of a statement*)
Given a PAL $\mathcal{A} = (S, A, R, P)$, the probability of a statement $s \in \mathit{lit}(S)$ is defined as the sum of the probabilities of all rule selections $R_s \in \mathcal{P}(R)$ in which s is acceptable.

4.2 Transformation to CHRiSM

We now introduce a transformation from an arbitrary probabilistic argumentation logic $\mathcal{A} = (S, A, R, P)$ to a CHRiSM program $P_{\mathit{CHRiSM}}(\mathcal{A})$. The transformation generalizes the example of Section 3. We assume the list predicates `member`/2, `subset`/2, and `append`/2 (whose first argument is a list of lists) are defined in the host language (Prolog).

The transformed program starts with the same four rules as in Section 3. They ensure that `accept`/2 encodes an interpretation, redundant conditional statements are removed, as well as defeated statements. It ends with the rules for the `assume` phase as in Section 3.

In between, there are two CHRiSM rules for each rule of $\mathcal{A}$. Each rule $r \in R$:

$$s_1 \wedge \ldots \wedge s_n \Rightarrow c_1 \wedge \ldots \wedge c_m \text{ assuming } a_1 \wedge \ldots \wedge a_k$$

is transformed into one simple probabilistic CHRiSM rule:

```
P(r) ?? begin ==> selected(r).
```

and one non-probabilistic CHRiSM rule:

```
1 ?? selected(r), accept(s1,C1), ..., accept(sn,Cn)
   ==> append([C1, ..., Cn, [a1, ..., ak] ], NC),
          accept(c1,NC), ..., accept(cm,NC).
```

where all literals of the form $\neg x$ are encoded as `not(`x`)`.

In the example transformation of Section 3, we have combined the selection and application rules into one probabilistic rule by "unfolding" the `selected`/1 constraint. This is not sound in general: if a rule can have several instances, or if the conditional statements in its left hand side can have their condition simplified (e.g. in the `assume` phase), the rule should either never fire, or always fire, depending on whether it was selected or not. For PAL rules where the left hand side statements can only be inferred unconditionally (in particular, for rules without a left hand side, i.e. facts), it is sound to do this "unfolding". This is the case in the mad cow example.

Correctness. It is relatively straightforward to see that when the `assume` phase starts, `accept`/2 encodes an interpretation that satisfies the criterion of Def. 4.5 w.r.t. the rule selection encoded by `selected`/1. It also does not contain gratuitous statements. However, the interpretation is not necessarily minimal. The `assume` phase searches for minimal interpretations by nondeterministically relaxing the assumptions. Since the relaxed `accept`/2 constraints will cause the transformed PAL rules to be revisited, the criterion of Def. 4.5 remains satisfied.

Advantages and disadvantages of CHRiSM. We could also attempt to implement PALs in other probabilistic logic formalisms, like LPAD (Vennekens, Verbaeten et al. 2004), PRISM (Sato 2008), or ProbLog (Kimmig, Demoen et al. 2011). One approach would be to implement assumptions using negation-as-failure. For example, the mad cow example could be expressed as an LPAD program as follows:

```
c :- r1, a, b, \+ not_app_r1.          r1.      a.   b:0.9.
not_app_r1 :- r2, d.                   r2:0.8.       d:0.5.
not_app_r1 :- r3, e, \+ not_app_r3.    r3:0.5.       e:0.2.
not_app_r3 :- r4, f.                   r4:0.5.       f:0.3.
```

and in this simple example, that would work. However, in the general case, explicit conditions are needed. Consider for example the tux example. In CHRiSM, we can use the constraint store to keep track of explicit conditions, and use multi-headed rules to simplify redundant conditions, retract undercut statements, and detect contradictions. Combined with standard Prolog search (in the `assume` rule), this gives us the expressivity needed to implement PALs in a straightforward way.

The downside of our CHRiSM implementation of PALs is that for probability computations (and for learning), we are basically generating all possible worlds (rule selections) and construct full interpretations for each world in a bottom-up (forward chaining) manner. This approach clearly does not scale well to PAL programs with a large amount of probabilistic rules, since the number of possible worlds is exponential in the number of probabilistic rules. In the case of legal reasoning, we do not think this is a major concern: the total number of rules and claims relevant to the case should be rather small in practice (after all, at some point all relevant information has to fit in the mind of the judge, who is only human). Moreover, only some of the rules and claims will have to be considered as being probabilistic — the others can be safely assigned a probability of 1 (or 0), so they do not contribute to the exponential blowup.

5 Learning

For now, we have assumed the probabilities to be known in advance. As mentioned in the conclusion of (Riveret, Rotolo et al. 2007), an issue is: *where do these numbers come from?* They suggest a statistical analysis of known precedents. The CHRiSM framework gives us exactly the tools needed to do this.

Instead of using fixed probabilities, we can make some or all probabilities learnable. We can then use a "training set" consisting of the outcomes of earlier similar cases — we call these the *observations* — to find a maximum likelihood probability distribution to fit the observations.

Some of the observations may be *full* observations, meaning that we not only know

Rule	*Original probability*	*Learned probability*
`a`	1	0.999999723
`b`	0.9	0.875267302
`d`	0.5	0.540243696
`e`	0.2	0.204545455
`f`	0.3	0.258644714
`r1`	1	0.999975929
`r2`	0.8	0.723429684
`r3`	0.5	0.547720330
`r4`	0.5	0.503165649

Table 1. *Re-discovering the probabilities with CHRiSM's learning algorithm.*

the final outcome, but also how exactly the reasoning went: what statements were put forward, what statements were accepted or rejected. More realistically, we only have *partial* observations: e.g. the final outcome is known, but not the intermediate steps. In CHRiSM we can use both.

As a proof of concept, let us try to "rediscover" the original probabilities in our running example. Assume for the sake of the example we have a database of 1100 prior rulings, but we do not have time to read through all of them to find out exactly what the reasoning was. Say we take 100 random samples and input them as full observations, like this:

```
begin <==> accept(not app(r1),[]),accept(d,[]),accept(b,[]),accept(a,[]).
begin <==> accept(d,[]),accept(a,[]).
begin <==> accept(c,[]),accept(b,[]),accept(a,[]),accept(app(r1),[]).
```

For example, in the first case, even though both `a` and `b` were accepted, `c` was not, either because rule `r1` was not applied – remember, we do not know its probability – or because its assumption `app(r1)` was refuted, i.e., `d` was accepted and rule `r2` was applied.
The remaining 1000 cases are not studied in that much depth: all that was checked is who won (i.e. was `c` accepted?) and whether or not statement `e` ("the cow was mad") was accepted. For the four possible combinations, the following counts were recorded:

```
 62 times begin ===> accept(c,[]), accept(e,[]).
432 times begin ===> accept(c,[]), ~accept(e,[]).
138 times begin ===> ~accept(c,[]), accept(e,[]).
368 times begin ===> ~accept(c,[]), ~accept(e,[]).
```

In full observations, the right hand side of the large double arrow has to be exhaustive, so there is no need for explicit negation-as-absence. In partial observations, explicit negation (denoted by tilde) can be useful, like in the above example.

The full observations above were obtained by simply taking 100 random samples (i.e. running the query `sample begin`) on the original program with the explicit probabilities; the counts for the partial observations were obtained by computing the probabilities for each of the four cases and multiplying them by 1000.

Now that we have a training set, we can use the built-in learning algorithm to find a probability distribution that fits the data. The resulting probabilities are shown in Table 1; they approximate the original probabilities reasonably well.

6 Conclusion

We have defined probabilistic argumentation logic (PAL) and showed how it can be used for probabilistic legal reasoning in the style of Riveret, Rotolo et al. (2007). We provided an implementation of PAL in CHRiSM through a straightforward encoding. Where previous work only showed by hand-performed examples how the problems could be solved, we provide a formalism and its encoding in CHRiSM that implements the problems and automates the generation of solutions. This automation goes much further than the original problem statement. The resulting CHRiSM program can be used to compute the probabilities of the possible outcomes, to obtain random samples, and to learn some or all underlying probabilities, solving an open problem of (Riveret, Rotolo et al. 2007). We have presented only an academic proof-of-concept of the learning method; it would be an interesting topic for future work to apply this work to real-world data.

For reasons of space, we cannot elaborate on the relationships between PAL and all the existing proposals for defeasible reasoning, argumentation, abduction, and non-monotonic logic in general. Preliminary results (Sneyers, De Schreye et al. 2013) indicate that Nute (2001)'s defeasible logic corresponds to a fragment of PAL. We developed a transformation from defeasible logic to PAL that is sound and weakly complete. In future work, the expressiveness of PAL has to be compared to that of other formalisms.

References

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *A.I. 77,* 2, 321–358.

Frühwirth, T. 2009. *Constraint Handling Rules.* Cambridge University Press.

Frühwirth, T. and Raiser, F., Eds. 2011. *Constraint Handling Rules: Compilation, Execution, and Analysis.* BOD.

Kakas, A. C., Kowalski, R. A., and Toni, F. 1992. Abductive logic programming. *Journal of logic and computation 2,* 6, 719–770.

Kimmig, A., Demoen, B., De Raedt, L., et al. 2011. On the implementation of the probabilistic logic programming language ProbLog. *TPLP 11,* 2-3, 235–262.

Nute, D. 2001. Defeasible logic: theory, implementation, and applications. In *Proceedings of INAP 2001, 14th International Conference on Applications of Prolog.* 87–114.

Prakken, H. and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics 7,* 1, 25–75.

Riveret, R., Rotolo, A., Sartor, G., Prakken, H., and Roth, B. 2007. Success chances in argument games: a probabilistic approach to legal disputes. In *JURIX.* Vol. 165. 99–108.

Roth, B., Riveret, R., Rotolo, A., and Governatori, G. 2007. Strategic argumentation: a game theoretical investigation. In *ICAIL.* ACM, 81–90.

Sato, T. 2008. A glimpse of symbolic-statistical modeling by PRISM. *Journal of Intelligent Information Systems 31*, 161–176.

Sneyers, J., De Schreye, D., and Frühwirth, T. 2013. CHRiSM and probabilistic argumentation logic. In *CHR 2013, 10th International Workshop on Constraint Handling Rules.*

Sneyers, J., Meert, W., Vennekens, J., Kameya, Y., and Sato, T. 2010. CHR(PRISM)-based probabilistic logic learning. *TPLP 10,* 4-6, 433–447.

Sneyers, J., Van Weert, P., Schrijvers, T., and De Koninck, L. 2010. As time goes by: Constraint Handling Rules. *TPLP 10,* 1, 1–47.

Vennekens, J., Verbaeten, S., and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *ICLP 2004.* LNCS, vol. 3132. Springer, 431–445.