

Probabilistic Termination of CHRiSM Programs

Jon Sneyers* and Danny De Schreye

Dept. of Computer Science, K.U.Leuven, Belgium
{jon.sneyers,danny.deschreye}@cs.kuleuven.be

Abstract. Termination analysis has received considerable attention in Logic Programming for several decades. In recent years, probabilistic extensions of Logic Programming languages have become increasingly important. Languages like PRISM, CP-Logic, ProbLog and CHRiSM have been introduced and proved very useful for addressing problems in which a combination of logical and probabilistic reasoning is required. As far as we know, the termination of probabilistic logical programs has not received any attention in the community so far.

Termination of a probabilistic program is not a crisp notion. Given a query, such a program does not simply either terminate or not terminate, but it terminates with a certain probability.

In this paper, we explore this problem in the context of CHRiSM, a probabilistic extension of CHR. We formally introduce the notion of probabilistic termination. We study this concept on the basis of a number of case studies. We provide some initial sufficient conditions to characterize probabilistically terminating programs and queries. We also discuss some challenging examples that reveal the complexity and interest of more general settings. The paper is intended as a first step in a challenging and important new area in the analysis of Logic Programs.

Keywords: Termination Analysis, Probabilistic LP, Constraint Handling Rules.

1 Introduction

Termination analysis has received considerable attention from the Logic Programming research community. Over several decades, many key concepts have been defined and formally studied (e.g. acceptability [3] and various variants of it), several powerful techniques for automatic verification of termination have been designed (e.g. the query-mapping pairs [15], constraint-based termination [7], the dependency-pairs framework [20]) and systems, implementing these techniques, currently provide very refined analysis tools for the termination property (e.g. cTI [17], Aprove [12], Polytool [21]). The main goals have been to support the study of total correctness of programs, to facilitate debugging tasks and to provide termination information for program optimization techniques such as partial evaluation and other transformation systems [16, 14, 29].

* This research is supported by F.W.O. Flanders.

Several Logic Programming related languages have been considered in all this work. Most research has addressed pure Prolog, but other languages, such as full Prolog [25, 28], CLP [18] and CHR [8, 30, 22] have also been considered.

In the past decade, probabilistic extensions of Logic Programming have become increasingly more important. Languages like PRISM [24] and ProbLog [6] extend Logic Programming with probabilistic reasoning and allow to tackle applications that require combinations of logical and probabilistic inference. Probabilistic termination analysis for imperative languages [19, 13] and for rewrite systems [5, 4] has already been studied in the past. However, as far as we know, termination analysis in the context of probabilistic logic programming has not yet received any attention, with the exception of some comments on probabilistic termination in [10]. It is the aim of the current paper to provide an initial investigation of the problem, mostly based on case studies.

Our study will be performed in the context of the probabilistic-logical language CHRiSM [26], a language based on CHR [9, 27, 11] and PRISM [24].

CHR – Constraint Handling Rules – is a high-level language extension based on multi-headed rules. Originally, CHR was designed as a special-purpose language to implement constraint solvers (see e.g. [2]), but in recent years it has matured into a general purpose programming language. 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). For instance, several CHR(Prolog) systems are available [27].

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

In [26], a new formalism was introduced, called CHRiSM, short for CHance Rules Induce Statistical Models. It is based on CHR(PRISM) and it combines the advantages of CHR and those of PRISM.

By way of motivation, let us consider a simple example of a CHRiSM program and briefly look at its termination properties.

Example 1 (Repeated Coin Flipping) *Suppose we flip a fair coin, and if the result is `tail`, we stop, but if it is `head`, we flip the coin again. We can model this process in CHRiSM as follows:*

```
flip <=> head:0.5 ; tail:0.5.  
head <=> flip.
```

In Section 2 we formally introduce the syntax and semantics of CHRiSM, but intuitively, the first rule states that the outcome of a `flip` event has an equal probability of 0.5 to result in `head` or in `tail`. The second rule is not probabilistic (or has probability 1): if there is `head`, we always `flip` again.

The program would typically be activated by the query `flip`. In a computation for the query `flip`, the first rule will either select `head` or `tail`, with equal probability. The choice is not backtrackable.

It is unclear whether we should consider this program as terminating or non-terminating. The program has an infinite derivation in which the coin always lands on **head**. However, out of all the possible derivations, there is only one infinite one and its probability of being executed is $\lim_{n \rightarrow \infty} (0.5)^n = 0$. So, if we execute this program, the probability of it terminating is equal to 1. Therefore, it is debatable whether we should call the program non-terminating and we will need the more refined notion of probabilistic termination. \square

In this paper we will study the notion of probabilistic termination. Most of this study is based on a number of case studies, where we compute the probabilities of termination for specific programs. We also compute the expected number of rule applications for some of these programs. For some classes of programs, we are able to generalize the results of our examples and formulate and prove probabilistic termination theorems. However, for the more complex programs, our study reveals that the mathematical equations modelling the probability of termination sometimes become too complex to solve.

The paper is organized as follows. In Section 2 we recall the syntax and semantics of CHRiSM. In Section 3 we define probabilistic termination of a CHRiSM program. We relate it to universal termination and we study some simple examples. Section 4 introduces a class of programs that behave like Markov chains, for which we provide a termination criterion. In Section 5 we discuss more complex examples and show that solving such examples provides a challenge for currently available mathematical techniques. We conclude in Section 6.

2 CHRiSM

In this section we briefly recall the CHRiSM programming language, in order to make this paper as self-contained as possible given the limited space. However we encourage the reader to refer to [26] for a more detailed description.

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 $c(\mathbf{t}_1, \dots, \mathbf{t}_n)$ looks like a Prolog predicate: it has a functor c of some arity n and arguments $\mathbf{t}_1, \dots, \mathbf{t}_n$ 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*.

We use $\{\!\!\{ \}$ to denote multisets, \uplus for multiset union, \subseteq for multiset subset, and $\exists_A B$ to denote $\exists x_1, \dots, x_n : B$, with $\{x_1, \dots, x_n\} = \text{vars}(B) \setminus \text{vars}(A)$, where $\text{vars}(A)$ are the (free) variables in A ; if A is omitted it is empty.

Chance rules. A chance rule has the following form: $P \text{ ?? } Hk \setminus Hr \Leftrightarrow G \mid B$, where P is a probability expression (as defined below), Hk is a conjunction of (kept head) constraints, Hr is a conjunction of (removed head) constraints, G is a guard condition, 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, written as “ $P \text{ ?? } Hk \implies G \mid B$ ”. If both Hk and Hr are non-empty, the rule is called a *simpagation* rule. The guard G is optional; if it is omitted, the “ \mid ” is also omitted. The body B is a conjunction of CHRiSM constraints, Prolog goals, and probabilistic disjunctions (as defined below).

Intuitively, the meaning of a chance rule is as follows: If the constraint store \mathbb{S} contains elements that match the head of the rule (i.e. if there is a (satisfiable) matching substitution θ such that $(\theta(Hk) \uplus \theta(Hr)) \subseteq \mathbb{S}$), and furthermore, the guard $\theta(G)$ 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 the probability expression P , the rule instance is either ignored or it actually leads to a rule application. Every rule instance may only be considered once.

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.

Probability expressions. In this paper, we assume probabilities to be fixed numbers. The CHRiSM system also supports learnable probabilities and other types of probability expressions. We refer to [26] for an overview.

Probabilistic disjunction. The body B of a CHRiSM rule may contain probabilistic disjunctions: “ $D1:P1 \ ; \ \dots \ ; \ Dn:Pn$ ” indicates that a disjunct Di is chosen with probability Pi . The probabilities should sum to 1 (otherwise a compile-time error occurs). Unlike CHR^\vee disjunctions [1], which create a choice point, probabilistic disjunctions are *committed-choice*: once a disjunct is chosen, the choice is not undone later. However, when later on in a derivation, the same disjunction is reached again, the choice can of course be different.

Operational Semantics. The operational semantics of a CHRiSM program \mathcal{P} is given by a state-transition system that resembles the abstract operational semantics ω_t of CHR [27]. The execution states are defined analogously, except that we additionally define a unique failed execution state, which is denoted by “*fail*” (because we don’t want to distinguish between different failed states). We use the symbol $\omega_t^{??}$ to refer to the (abstract) operational semantics of CHRiSM.

Definition 1 (identifiers). An identified constraint $c\#i$ is a CHRiSM constraint c associated with some unique integer i . This number serves to differentiate between copies of the same constraint. We introduce the functions $\text{chr}(c\#i) = c$ and $\text{id}(c\#i) = i$, and extend them to sequences and sets, e.g.:

$$\text{chr}(S) = \{\{c\#i \in S\}\}$$

Definition 2 (execution state). An execution state σ is a tuple $\langle \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n$. The goal \mathbb{G} is a multiset of constraints to be rewritten to solved form. The store \mathbb{S} is a set of identified constraints that can be matched with rules in the program \mathcal{P} . Note that $\text{chr}(\mathbb{S})$ is a multiset although \mathbb{S} is a set. The built-in store \mathbb{B} is the conjunction of all Prolog goals that have been called so far. The history \mathbb{T} is a

1. **Fail.** $\langle \{b\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{1}{\mathcal{P}}} fail$
 where b is a built-in (Prolog) constraint and $\mathcal{D}_{\mathcal{H}} \models \neg \exists (\mathbb{B} \wedge b)$.
2. **Solve.** $\langle \{b\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{1}{\mathcal{P}}} \langle \mathbb{G}, \mathbb{S}, b \wedge \mathbb{B}, \mathbb{T} \rangle_n$
 where b is a built-in (Prolog) constraint and $\mathcal{D}_{\mathcal{H}} \models \exists (\mathbb{B} \wedge b)$.
3. **Introduce.** $\langle \{c\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{1}{\mathcal{P}}} \langle \mathbb{G}, \{c\#n\} \cup \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_{n+1}$
 where c is a CHRiSM constraint.
4. **Probabilistic-Choice.** $\langle \{d\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{p_i}{\mathcal{P}}} \langle \{d_i\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n$
 where d is a probabilistic disjunction of the form $d_1:p_1 ; \dots ; d_k:p_k$ or of the form $P \text{ ?? } d_1 ; \dots ; d_k$, where the probability distribution given by P assigns the probability p_i to the disjunct d_i .
5. **Maybe-Apply.** $\langle \mathbb{G}, H_1 \uplus H_2 \uplus \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{1-p}{\mathcal{P}}} \langle \mathbb{G}, H_1 \uplus H_2 \uplus \mathbb{S}, \mathbb{B}, \mathbb{T} \cup \{h\} \rangle_n$
 $\langle \mathbb{G}, H_1 \uplus H_2 \uplus \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n \xrightarrow{\frac{p}{\mathcal{P}}} \langle B \uplus \mathbb{G}, H_1 \uplus \mathbb{S}, \theta \wedge \mathbb{B}, \mathbb{T} \cup \{h\} \rangle_n$
 where the r -th rule of \mathcal{P} is of the form $P \text{ ?? } H'_1 \setminus H'_2 \Leftarrow G \mid B$,
 θ is a matching substitution such that $chr(H_1) = \theta(H'_1)$ and $chr(H_2) = \theta(H'_2)$,
 $h = (r, id(H_1), id(H_2)) \notin \mathbb{T}$, and $\mathcal{D}_{\mathcal{H}} \models \mathbb{B} \rightarrow \exists_{\mathbb{B}}(\theta \wedge G)$. If P is a number, then $p = P$. Otherwise p is the probability assigned to the success branch of P .

Fig. 1. Transition relation $\xrightarrow{\mathcal{P}}$ of the abstract operational semantics $\omega_i^{??}$ of CHRiSM.

set of tuples, each recording the identifiers of the CHRiSM constraints that fired a rule and the rule number. The history is used to prevent trivial non-termination: a rule instance is allowed to be considered only once. Finally, the counter $n \in \mathbb{N}$ represents the next free identifier.

We use $\sigma, \sigma_0, \sigma_1, \dots$ to denote execution states and Σ to denote the set of all execution states. We use $\mathcal{D}_{\mathcal{H}}$ to denote the theory defining the host language (Prolog) built-ins and predicates used in the CHRiSM program. For a given program \mathcal{P} , the transitions are defined by the binary relation $\xrightarrow{\mathcal{P}} \subset \Sigma \times \Sigma$ shown in Figure 1. Every transition is annotated with a probability.

Execution of a query Q proceeds by exhaustively applying the transition rules, starting from an initial state (root) of the form $\sigma_Q = \langle Q, \emptyset, true, \emptyset \rangle_0$ and performing a random walk in the directed acyclic graph defined by the transition relation $\xrightarrow{\mathcal{P}}$, until a leaf node is reached, which is called a final state. We use Σ_f to denote the set of final states. The probability of a path from an initial state to the state σ is simply the product of the probabilities along the path.

We assume a given execution strategy ξ that fixes the non-probabilistic choices in case multiple transitions are applicable (see section 4.1 of [26]).

We use $\sigma_0 \xrightarrow{\mathcal{P}}^* \sigma_f$ to denote all k different derivations from σ_0 to σ_f :

$$\begin{array}{c}
 \sigma_0 \xrightarrow{\frac{p_{1,1}}{\mathcal{P}}} \sigma_{1,1} \xrightarrow{\frac{p_{1,2}}{\mathcal{P}}} \sigma_{1,2} \xrightarrow{\frac{p_{1,3}}{\mathcal{P}}} \dots \xrightarrow{\frac{p_{1,l_1}}{\mathcal{P}}} \sigma_f \\
 \vdots \\
 \sigma_0 \xrightarrow{\frac{p_{k,1}}{\mathcal{P}}} \sigma_{k,1} \xrightarrow{\frac{p_{k,2}}{\mathcal{P}}} \sigma_{k,2} \xrightarrow{\frac{p_{k,3}}{\mathcal{P}}} \dots \xrightarrow{\frac{p_{k,l_k}}{\mathcal{P}}} \sigma_f
 \end{array}$$

where

$$p = \sum_{i=1}^k \prod_{j=1}^{l_i} p_{i,j}.$$

If σ_0 is an initial state and σ_k is a final state, then we call these derivations an *explanation set* with total probability p for the query σ_0 and the result σ_k . Note that if $k = 0$, i.e. there is no derivation from σ_0 to σ_f , then $p = 0$. We define a function $prob$ to give the probability of an explanation set: $prob(\sigma_0 \xrightarrow{\mathcal{P}}^* \sigma_k) = p$.

If σ_0 is an initial state and there exist infinite sequences s_i of transitions

$$\sigma_0 \xrightarrow{\mathcal{P}}^{p_{i,1}} \sigma_{i,1} \xrightarrow{\mathcal{P}}^{p_{i,2}} \sigma_{i,2} \xrightarrow{\mathcal{P}}^{p_{i,3}} \dots$$

then we call these sequences infinite derivations from σ_0 . We use $\sigma_0 \xrightarrow{\mathcal{P}}^* \infty$ to denote the (possibly infinite) set D of infinite derivations from σ_0 , where

$$p = \lim_{l \rightarrow \infty} \sum_{i=1}^{|D|} \prod_{j=1}^l p_{i,j}.$$

Note that if all rule probabilities are 1 and the program contains no probabilistic disjunctions — 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.

3 Probabilistic Termination

In the contexts of Prolog and CHR, the usual notion of termination is *universal termination*: a program \mathcal{P} terminates for a query Q if Q does not have an infinite derivation for \mathcal{P} .

For some CHRiSM programs, universal termination is too strong. In order to be able to execute (sample) a program, *probabilistic termination* is sufficient.

Definition 3 (Probabilistic termination). *A program \mathcal{P} probabilistically terminates for a query Q with probability p if the probability of the event that the computation for Q in \mathcal{P} halts is equal to p , i.e.*

$$p = \sum_{\sigma \in \Sigma_f} prob(\sigma_Q \xrightarrow{\mathcal{P}}^* \sigma) = 1 - prob(\sigma_Q \xrightarrow{\mathcal{P}}^* \infty).$$

A program \mathcal{P} probabilistically terminates for a query Q if the program probabilistically terminates for Q with probability 1. A program \mathcal{P} probabilistically terminates if it probabilistically terminates for all finite queries.

Note that the above definition does not give a general practical method to compute the termination probability for a given query, and like universal termination, probabilistic termination is undecidable. Note also that in general the number of finite derivations may be infinite.

Although many CHRiSM programs do not universally terminate (so they have infinite derivations) universal termination is of course sufficient for probabilistic

termination. This already provides us with a practical way of proving probabilistic termination of one class of CHRiSM programs. In general, we can associate to any CHRiSM program \mathcal{P} a corresponding CHR^\vee program [1], $\text{CHR}^\vee(\mathcal{P})$, by removing all the probability information from \mathcal{P} . As already mentioned, apart from removing the probability factors, this transformation changes committed choice disjunctions into backtrackable disjunctions. However, from a perspective of proving universal termination, this is not a problem, because we need to prove that all derivations are finite anyway.

Proposition 1. *For any CHRiSM program \mathcal{P} and query Q , \mathcal{P} is probabilistically terminating for Q if $\text{CHR}^\vee(\mathcal{P})$ is universally terminating for Q .*

Proof. If $\text{CHR}^\vee(\mathcal{P})$ has no infinite derivation for Q , then \mathcal{P} has no infinite derivation for Q . Thus the probability of halting is one.

Proving that $\text{CHR}^\vee(\mathcal{P})$ universally terminates for Q can be done using the techniques presented in [8], [30] or [22]. The latter technique has been automated and implemented [23]. Of course, these techniques were developed for CHR, rather than for CHR^\vee , but again, a CHR^\vee program can easily be transformed into a CHR program with the same universal termination behavior.

Example 2 (Low-power Countdown) *Consider the CHRiSM program:*

```
0.9 ?? countdown(N) <=> N > 1 | countdown(N-1).
0.9 ?? countdown(0) <=> writeln('Happy New Year!').
```

representing a New Year countdown device with low battery power, which may or may not display its New Year wishes starting from a query `countdown(10)`. At every tick, there is a 10% chance that the battery dies and the countdown stops.

Consider the CHR program obtained by omitting the probabilities. The ranking techniques of all three of the approaches presented in [8, 30, 22], prove universal termination for that program and the same query. Thus, Low-power New Years Countdown will terminate (universally and probabilistically) as well. \square

However, the main attention in this paper will be addressed to the case in which the CHRiSM program does not universally terminate.

Example 3 (Repeated Coin Flipping cont'd) *Consider again Example 1.*

```
flip <=> head:0.5 ; tail:0.5.
head <=> flip.
```

Recall that the only infinite derivation has probability $\lim_{n \rightarrow \infty} (0.5)^n = 0$, so that we can conclude that the program probabilistically terminates.

Figure 2 shows the derivation tree for this program, assuming the query `flip`. This derivation tree corresponds to a finite cyclic derivation graph. There is only one cycle in this graph and its probability is less than one. \square

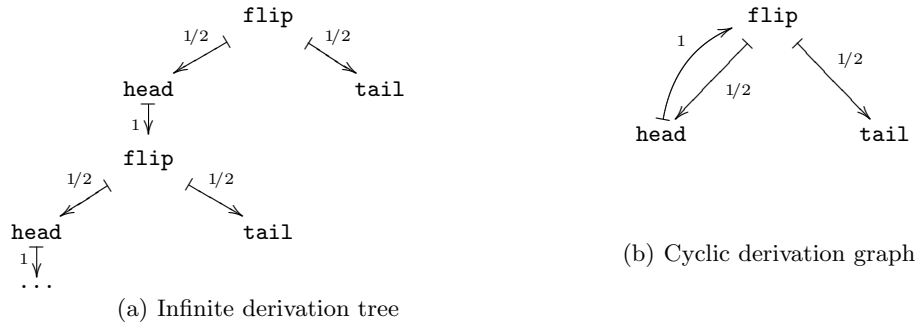


Fig. 2. Repeated coin flipping example.

Example 4 (Basic a-to-a) *In terms of termination behavior, the coin flipping program is equivalent to the following program, given the query a :*

$0.5 \text{ ?? } a \Leftarrow a.$

To analyze the probabilistic termination of programs we have to compute the termination probability as the sum of the probabilities of all terminating derivations. It is not difficult to see that for the above program, just as in the original coin flipping program, the termination probability is $\sum_{i=1}^{\infty} (0.5)^i = 1$. \square

The above example can be generalized to the case where the probability is some p between 0 and 1, instead of 0.5. Consider the rule “ $p \text{ ?? } a \Leftarrow a.$ ” where p is a fixed probability. All terminating derivations consist of n rule applications followed by one non-applied rule instance. Thus, the probability of such a derivation is $p^n(1-p)$. Hence, the total termination probability s for the above program is $s = \sum_{n=0}^{\infty} p^n(1-p)$. To solve this infinite sum it suffices to note that $s - ps = 1 - p$, so if $p < 1$ we have $s = (1-p)/(1-p) = 1$. If $p = 1$ we get $s = 0$ since every term is zero.

We can compute the expected number of rule applications (i.e., the average run time) as follows, assuming $p < 1$ so the probability of non-termination is zero: $\sum_{n=0}^{\infty} p^n(1-p)n = p/(1-p)$. So e.g. if $p = 3/4$, then the expected number of rule applications is 3.

4 Markov Chains and MC-type computations

Example 5 (Bull-bear) *Consider the following CHRiSM program, which implements a Markov chain modeling the evolution of some market economy:*

```
s(T,bull) ==> (s(T+1,bull):0.9 ; s(T+1,bear):0.1).
s(T,bear) ==> (s(T+1,bear):0.7 ; s(T+1,bull):0.2 ; s(T+1,recession):0.1).
s(T,recession) ==> (s(T+1,recession):0.7 ; s(T+1,bear):0.2 ; stop:0.1).
```

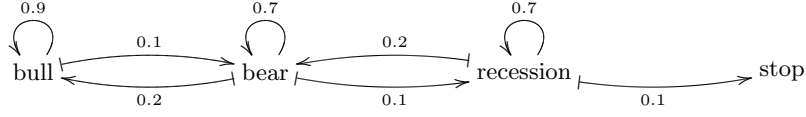


Fig. 3. Markov chain representing the economy.

Figure 3 illustrates the transitions. In every state the most likely option is to stay in that state. Also, a bull market can become a bear market, a bear market can recover to a bull market or worsen into a recession, and in a recession we can recover to a bear market or the market economy transition system may come to an end (e.g. a socialist revolution happened).

In this case, the program terminates probabilistically. This can be shown as follows. From every execution state, the probability of termination has a nonzero lower bound, as can be verified from Fig.3. Indeed, it is easy to see that in every state, the probability of terminating “immediately” (i.e. by taking the shortest path to the final state “stop”) is at least 0.001. Now, every infinite derivation has to visit one of the states in Fig.3 infinitely often. Let p_x be the total probability of all derivations that visit state x infinitely often, then the probability of non-termination is at most $p_{\text{bull}} + p_{\text{bear}} + p_{\text{recession}}$. We now show that $p_x = 0$. Consider all subderivations of the form $x \rightarrow y_1 \rightarrow y_2 \rightarrow \dots \rightarrow x$ where all intermediate $y_i \neq x$. The total probability for all such subderivations has to be less than 0.999, since there is a subderivation $x \xrightarrow{p}^* \text{stop}$ with $p \geq 0.001$ (where all intermediate steps are different from x). This means that the probability p_x is bounded from above by

$$\lim_{n \rightarrow \infty} (0.999)^n = 0$$

so the probability of termination is one. \square

In the previous two examples, the transition graph is essentially finite, in the following sense: there exists a finite set of abstractions of states and probabilistic transitions between these abstract states and an abstraction function from execution states to abstract states, such that for all reachable executions states, concrete transitions between these states are mapped to transitions between abstract states, with the same probability, and conversely.

More formally, we introduce MC-graphs and MC-type computations.

Definition 4 (MC-graph). An MC-graph is an annotated, directed graph, (V, A, L) , consisting of a finite set of vertices $V = \{v_1, \dots, v_n\}$, a set of arcs, $A \subseteq V \times V$, and a function $L : A \rightarrow]0, 1]$. We refer to L as the probability labeling for A .

It should be clear that an MC-graph represents a Markov Chain.

Definition 5 (MC-type computation). Let \mathcal{P} be a CHRiSM program and Q a query to \mathcal{P} . The computation for \mathcal{P} and Q consists of all possible transitions $\sigma \xrightarrow{\mathcal{P}} \sigma'$, reachable from the initial state $\langle Q, \emptyset, \text{true}, \emptyset \rangle_0$.

The computation for \mathcal{P} and Q is an MC-type computation, if there exists an MC-graph (V, A, L) and a function $\alpha : \Sigma \rightarrow V$, such that:

- If there is a transition $\sigma \xrightarrow{\frac{p}{p}} \sigma'$ of type **Probabilistic-Choice** or **Maybe-Apply** (see Fig. 1) in the computation for \mathcal{P} and Q , where $p > 0$ (we omit impossible transitions), then $(\alpha(\sigma), \alpha(\sigma')) \in A$ and $L((\alpha(\sigma), \alpha(\sigma'))) = p$.
- If $\alpha(\sigma) \in V$, $(\alpha(\sigma), v) \in A$ and $L((\alpha(\sigma), v)) = p$, then there exists a reachable execution state $\sigma' \in \Sigma$, such that $\alpha(\sigma') = v$ and $\sigma \xrightarrow{\frac{p}{p}} \sigma'$ is in the computation for \mathcal{P} and Q .

Example 6 (a-to-a, bull-bear cont'd) In the a-to-a example, $V = \{a, stop\}$, $A = \{(a, a), (a, stop)\}$, $L((a, a)) = p$, and $L((a, stop)) = 1 - p$. All reachable non-final execution states are mapped to a , all final states are mapped to $stop$.

In the bull-bear example, the MC-graph is essentially as represented in Figure 3, with the addition of the loop-arcs on the vertices bull, bear and recession. The mapping α maps an execution state of the form $\langle \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n$ to the node x iff $\mathbf{s}(-, \mathbf{x}) \in \mathbb{G}$ or $\mathbf{s}(-, \mathbf{x}) \# \mathbf{k} \in \mathbb{S}$ and $(-, k) \notin \mathbb{T}$. \square

Note that a leaf node (a node without outgoing edges) in the MC-graph corresponds to final states. We have the following criterion for probabilistic termination of MC-type computations.

Theorem 1. Let \mathcal{P} be a CHRiSM program and Q a query, such that the computation for \mathcal{P} and Q is MC-type and let (V, A, L) be the associated MC-graph. The program \mathcal{P} probabilistically terminates for Q if for every node $v_i \in V$, there is a path in A from v_i to a leaf node.

Proof. The argument is identical to the one for the bull-bear example above.

It is tempting (but wrong) to think that a CHRiSM program \mathcal{P} is probabilistically terminating if every cycle in its derivation graph has a probability $p < 1$. This is a tempting idea, because, for such programs, every infinite derivation has probability zero. However, even for MC-type computations, this is wrong.

Example 7 (Infinite Coin Flipping) The following program terminates with probability zero, although every infinite derivation has probability zero:

```
flip <=> head:0.5 ; tail:0.5.
head <=> flip.
tail <=> flip. □
```

5 The Drunk Guy on a Cliff and More

Now let us analyze a more challenging toy example in which the transition graph is essentially infinite. Consider the following rule:

```
0.5 ?? a <=> a, a.
```

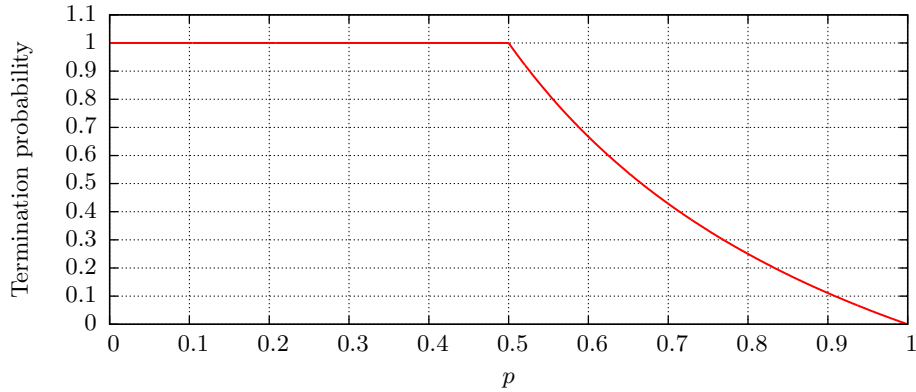


Fig. 4. Termination probability for the rule “p ?? a <=> a, a”.

We are trying to find the termination probability s for the query a . With probability 0.5, the program terminates immediately (the rule instance is not applied), and with probability 0.5, it terminates if and only if it (independently) terminates two times in a row for the query a . If terminating once has probability s , then terminating twice has probability s^2 , so we get the following equation:

$$s = 0.5 + 0.5s^2$$

The only solution to this equation is $s = 1$. Somewhat counter-intuitively, the above program does terminate probabilistically.

In general, consider the above rule with an arbitrary fixed probability p :

p ?? a <=> a, a.

We can compute the termination probability as follows. Again, either the program terminates immediately or it has to terminate twice from the query a , so $s = (1 - p) + ps^2$. Solving for s , we get $s = 1$ or $s = \frac{1-p}{p}$, so taking into account that $0 \leq s \leq 1$ (since s is a probability), we have $s = 1$ if $p \leq 1/2$ and $s = (1 - p)/p$ if $p \geq 1/2$ (see Fig. 4).

The “drunk guy on a cliff” puzzle is defined as follows. There is a sheer cliff, and a drunk guy is facing the cliff. He is staggering drunkenly back and forth. One single step forward from his current location will send him hurtling into the abyss, a step backward will bring him closer to safety. The chance of him staggering backwards (at any time) is p , the chance of him staggering forwards is $1 - p$. What is the chance that he will eventually fall into the abyss?

We can model the drunk guy on a cliff as follows:

```
dist(0) <=> true.
dist(N) <=> N > 0 | dist(N+1):p ; dist(N-1):(1-p).
```

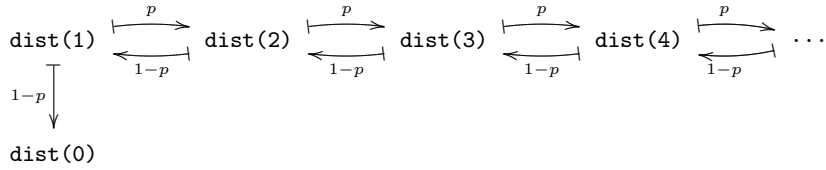


Fig. 5. Derivation graph for the drunk guy on a cliff program.

with the initial query `dist(1)`. The only way to terminate is by reaching `dist(0)`, i.e. by falling into the abyss. So the drunk guy on a cliff puzzle boils down to computing the termination probability of the above program.

For example, consider the case $p = 1/2$ and the query `dist(1)`. The probability of termination $s = s(\text{dist}(1))$ can be computed as follows:

$$s(\text{dist}(i)) = \frac{1}{2}s(\text{dist}(i-1)) + \frac{1}{2}s(\text{dist}(i+1))$$

Given that $s(\text{dist}(0)) = 1$, it is easy to verify that

$$s(\text{dist}(1)) = \frac{1}{2} + \frac{1}{2}s(\text{dist}(2)) = \frac{2}{3} + \frac{1}{3}s(\text{dist}(3)) = \frac{3}{4} + \frac{1}{4}s(\text{dist}(4)) = \dots = 1$$

It turns out that we have already solved this problem. Consider again the program consisting of the single rule “`p ?? a <=> a, a`”, and consider an $\omega_i^{??}$ execution state $\sigma = \langle \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n$ in a derivation starting from the query “`a`”. The probability of termination from state σ only depends on the number of `a/0` constraints for which the rule could still be applied, which is the following number: $|\mathbb{G} \uplus \{ \mathbf{a} \# n \in \mathbb{S} \mid (1, n) \notin \mathbb{T} \}|$. Let us call this number the *distance to termination* and denote it with $d(\sigma)$. When considering a rule instance, there are two options: with probability p , the distance increases by one (the rule is applied), and with probability $1-p$, the distance decreases by one (the rule is not applied). The program terminates as soon as $d(\sigma) = 0$.

An alternative way to compute the termination probability of the above programs is as follows. All terminating derivations consist of n rule instances that are applied (or distances that are increased) and $n+1$ rule instances that are not applied (or distances that are decreased), for some number n . For example, for $n = 3$ we have the following five derivations: `+++----`, `++-+---`, `+-++----`, `+-+-+---`, and `+--+----` where “`+`” means “applied” (or incremented) and “`-`” means “not applied” (or decremented). For a given number n of applied rule instances, the number of possible derivations is given by the n -th Catalan number¹ $C_n = (2n)!/(n!(n+1)!)$. This gives us the following

¹ The Catalan numbers are named after the Belgian mathematician Eugène Charles Catalan (1814-1894). They are sequence A000108 in The On-Line Encyclopedia of Integer Sequences (<http://oeis.org/A000108>).

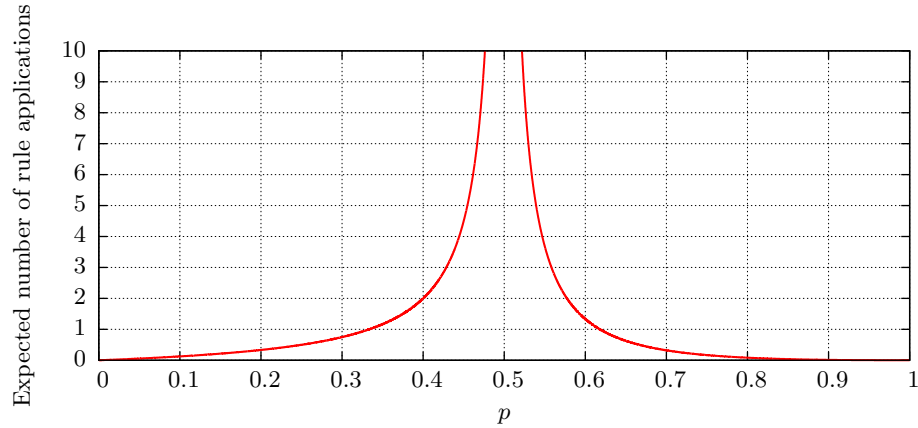


Fig. 6. Expected number of rule applications (if terminating) for “ $p \text{ ?? } a \Leftrightarrow a, a$ ”.

formula to compute the termination probability s :

$$s = \sum_{n=0}^{\infty} p^n (1-p)^{n+1} \frac{(2n)!}{n!(n+1)!}$$

The above infinite sum converges to the same values for s we already calculated above. However, this alternative formulation allows us to compute the expected number of rule applications (of the rule “ $p \text{ ?? } a \Leftrightarrow a, a$ ”). The expected number of rule applications is given by:

$$E_p = \sum_{n=0}^{\infty} p^n (1-p)^{n+1} \frac{(2n)!}{n!(n+1)!} n$$

Figure 6 shows the expected number of rule applications E_p as a function of p . As p gets closer to $1/2$, the value for E_p grows quickly: for $p = \frac{2^k - 1}{2^{k+1}}$, we have $E_p = \frac{2^k - 1}{2}$. The border case $p = 1/2$ is interesting: the termination probability is 1 but the expected number of rule applications is $+\infty$. As p gets larger than $1/2$, the termination probability drops and so does the expected number of rule applications of the (increasingly rare) terminating derivations.

Multi-headed rules. So far, we have only considered single-headed rules. One of the simplest “interesting” cases is the following rule: (with less than three a ’s in the body, the program terminates universally)

$$p \text{ ?? } a, a \Leftrightarrow a, a, a.$$

Given the query “ a, a ”, there are two rule instances to be considered: ($a\#1, a\#2$) and ($a\#2, a\#1$). If both instances are not applied (probability $(1-p)^2$), the program terminates; otherwise we are further away from termination.

We define the termination distance $d(\sigma)$ of an execution state $\sigma = \langle \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n$ as a pair $d(\sigma) = (n, m)$ where $n = |\mathbb{G}|$ is the number of “not yet considered” a’s and $m = |\mathbb{S}|$ is the number of “considered” a’s. Given the query “a, a”, the initial state has distance $(2, 0)$.

The termination probability $s(n, m)$ of a state with distance (n, m) can be computed as follows. If $n = 0$, the rule can no longer be applied so we have termination. If $n > 0$, we can take one a/0 and consider all matching rule instances. Since there are m possible partner constraints and two (symmetric) occurrences of the active constraint, there are $2m$ rule instances to consider. If none of these rule instances are applied, we just add the a/0 constraint to the store, so the new distance will be $(n - 1, m + 1)$. However, if one of the rule instances is applied, we get a new distance $(n + 2, m - 1)$. So the termination probability $s(n, m)$ is given by the following equations:

$$\begin{cases} s(0, m) = 1 \\ s(n, m) = (1 - p)^{2m} s(n - 1, m + 1) + (1 - (1 - p)^{2m}) s(n + 2, m - 1) \end{cases}$$

Note that if $m = 0$, there are no partner constraints so there are no rule instances to consider, i.e. $s(n, 0) = s(n - 1, 1)$.

Unfortunately, we have not found a closed-form solution for the above equations. The example shows that, while the complexity of the programs increases, the mathematical models representing the probability of termination become too complex to be solved by standard techniques.

6 Conclusion and Future Work

In this paper we presented the results of an initial investigation of the concept of probabilistic termination of CHRiSM programs. This research has mostly taken the form of a number of small case studies, in which we attempt to reveal the intuitions concerning the concept of probabilistic termination and present some ways of (manually) proving probabilistic termination. In the process, for some of the cases, we also study the expected number of rule applications.

For some classes of programs, we have generalised the observations in our case studies and we formulated and proved termination conditions. In particular, for universally termination programs we obviously also get probabilistic termination. Therefore, techniques developed to prove universal termination of CHR are sufficient to prove probabilistic termination of corresponding CHRiSM programs. We also identified the class of MC-type programs and formulated and proved a sufficient probabilistic termination condition for it.

For more general classes of programs, termination proofs may become quite complex. We elaborated on a few more complex (but still toy) cases, where sometimes we are able to solve the problem, but in other cases, we observe that the equations expressing probabilistic termination are too complex to be solved with standard techniques. In this exploratory paper we have focused on

intuition and examples. It would be interesting to investigate to what extent the theoretical work of [5, 4] can be transferred to our setting.

Finally, note that although this work has been in the context of CHRiSM, most of the ideas are also applicable to other probabilistic logic programming languages like PRISM and ProbLog. In most of the probabilistic inference algorithms used in the implementations of these languages (e.g. probability computation, learning), it is assumed that the program universally terminates. An interesting direction for future work is to generalize these algorithms such that they can handle (certain classes of) probabilistically terminating programs.

References

1. Abdennadher, S.: A language for experimenting with declarative paradigms. In: Frühwirth, T., et al. (eds.) RCoRP '00(bis) (Sep 2000)
2. Abdennadher, S., Rigotti, C.: Automatic generation of CHR constraint solvers. *TPLP* 5(4-5), 403–418 (2005)
3. Apt, K.R., Pedreschi, D.: Reasoning about termination of pure Prolog programs. *Inf. Comput.* 106(1), 109–157 (1993)
4. Bournez, O., Garnier, F.: Proving positive almost sure termination under strategies. In: Pfenning, F. (ed.) RTA. LNCS, vol. 4098, pp. 357–371. Springer (2006)
5. Bournez, O., Kirchner, C.: Probabilistic rewrite strategies: Applications to ELAN. In: Tison, S. (ed.) RTA, LNCS, vol. 2378, pp. 339–357. Springer (2002)
6. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: IJCAI. pp. 2462–2467 (2007)
7. Decorte, S., De Schreye, D., Vandecasteele, H.: Constraint-based automatic termination analysis of logic programs. *ACM TOPLAS* 21(6), 1137–1195 (1999)
8. Frühwirth, T.: Proving termination of constraint solver programs. In: New Trends in Constraints, Joint ERCIM/Compulog Net Workshop. pp. 298–317 (2000)
9. Frühwirth, T.: *Constraint Handling Rules*. Cambridge University Press (2009)
10. Frühwirth, T., Di Pierro, A., Wiklicky, H.: Probabilistic Constraint Handling Rules. In: Comini, M., Falaschi, M. (eds.) WFLP 2002. ENTCS 76, Elsevier (2002)
11. Frühwirth, T., Raiser, F. (eds.): *Constraint Handling Rules: Compilation, Execution, and Analysis* (March 2011)
12. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. *J. Autom. Reasoning* 37(3), 155–203 (2006)
13. Hurd, J.: A formal approach to probabilistic termination. In: Carreño, V., Muñoz, C., Tahar, S. (eds.) TPHOLs. LNCS, vol. 2410, pp. 230–245. Springer (2002)
14. Leuschel, M., Martens, B., De Schreye, D.: Controlling generalization and polyvariance in partial deduction of normal logic programs. *ACM TOPLAS* 20(1), 208–258 (1998)
15. Lindenstrauss, N., Sagiv, Y., Serebrenik, A.: Proving termination for logic programs by the query-mapping pairs approach. In: *Program Development in Computational Logic*. pp. 453–498. LNCS 3049 (2004)
16. Martens, B., De Schreye, D., Bruynooghe, M.: Sound and complete partial deduction with unfolding based on well-founded measures. In: FGCS. pp. 473–480 (1992)
17. Mesnard, F., Bagnara, R.: cTI: A constraint-based termination inference tool for ISO-Prolog. *TPLP* 5(1-2), 243–257 (2005)

18. Mesnard, F., Ruggieri, S.: On proving left termination of constraint logic programs. *ACM Trans. Comput. Log.* 4(2), 207–259 (2003)
19. Monniaux, D.: An abstract analysis of the probabilistic termination of programs. In: Cousot, P. (ed.) *SAS*. LNCS, vol. 2126, pp. 111–126. Springer (2001)
20. Nguyen, M.T., Giesl, J., Schneider-Kamp, P., De Schreye, D.: Termination analysis of logic programs based on dependency graphs. In: *Proc. LOPSTR '07*. pp. 8–22. LNCS 4915, Springer (2008)
21. Nguyen, M.T., De Schreye, D., et al.: Polytool: polynomial interpretations as a basis for termination analysis of logic programs. *TPLP* 11, 33–63 (2011)
22. Pilozzi, P., De Schreye, D.: Termination analysis of CHR revisited. In: *ICLP*. LNCS, vol. 5366, pp. 501–515. Springer (2008)
23. Pilozzi, P., De Schreye, D.: Automating termination proofs for CHR. In: *ICLP*. LNCS, vol. 5649, pp. 504–508. Springer (2009)
24. Sato, T.: A glimpse of symbolic-statistical modeling by PRISM. *Journal of Intelligent Information Systems* 31, 161–176 (2008)
25. Schneider-Kamp, P., Giesl, J., Ströder, T., et al.: Automated termination analysis for logic programs with cut. *TPLP* 10(4-6), 365–381 (2010)
26. Sneyers, J., Meert, W., Vennekens, J., Kameya, Y., Sato, T.: CHR(PRISM)-based probabilistic logic learning. *TPLP* 10(4-6) (2010)
27. Sneyers, J., Van Weert, P., Schrijvers, T., De Koninck, L.: As time goes by: Constraint Handling Rules — a survey of CHR research between 1998 and 2007. *TPLP* 10(1), 1–47 (January 2010)
28. Ströder, T., Schneider-Kamp, P., Giesl, J., et al.: A linear operational semantics for termination and complexity analysis of ISO Prolog. In: *LOPSTR* (2011), accepted
29. Vidal, G.: A hybrid approach to conjunctive partial deduction. In: *LOPSTR 2010, Revised Selected Papers*. LNCS, vol. 6564, pp. 200–214. Springer (2011)
30. Voets, D., De Schreye, D., Pilozzi, P.: A new approach to termination analysis of constraint handling rules. In: *LOPSTR*. pp. 28–42 (2008)