Open access • Journal Article • DOI:10.1017/S1471068409003767

Cp-logic: A language of causal probabilistic events and its relation to logic programming — Source link

Joost Vennekens, Marc Denecker, Maurice Bruynooghe

Institutions: Katholieke Universiteit Leuven

Published on: 01 May 2009 - Theory and Practice of Logic Programming (Cambridge University Press)

Topics: Probabilistic argumentation, Probabilistic CTL, Probabilistic logic, Probabilistic logic network and Autoepistemic logic

CP-logic: A Language of Causal Probabilistic Events and Its Relation to Logic Programming

JOOST VENNEKENS

MARC DENECKER

MAURICE BRUYNOOGHE

{joost, marcd, maurice}@cs.kuleuven.be
Dept. of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium

submitted July 2008; revised 6 January 2009; accepted 2 April 2009

Abstract

This papers develops a logical language for representing probabilistic causal laws. Our interest in such a language is twofold. First, it can be motivated as a fundamental study of the representation of causal knowledge. Causality has an inherent dynamic aspect, which has been studied at the semantical level by Shafer in his framework of probability trees. In such a dynamic context, where the evolution of a domain over time is considered, the idea of a causal law as something which guides this evolution is quite natural. In our formalization, a set of probabilistic causal laws can be used to represent a class of probability trees in a concise, flexible and modular way. In this way, our work extends Shafer's by offering a convenient logical representation for his semantical objects.

Second, this language also has relevance for the area of probabilistic logic programming. In particular, we prove that the formal semantics of a theory in our language can be equivalently defined as a probability distribution over the well-founded models of certain logic programs, rendering it formally quite similar to existing languages such as ICL or PRISM. Because we can motivate and explain our language in a completely self-contained way as a representation of probabilistic causal laws, this provides a new way of explaining the intuitions behind such probabilistic logic programs: we can say precisely which knowledge such a program expresses, in terms that are equally understandable by a non-logician. Moreover, we also obtain an additional piece of knowledge representation methodology for probabilistic logic programs, by showing how they can express probabilistic causal laws.

KEYWORDS: Uncertainty, Causality, Probabilistic Logic Programming

1 Introduction

Logic based languages, such as logic programming, play an important role in knowledge representation. One of the known weaknesses of such languages is that they are not well suited for representing probabilistic or uncertain knowledge. This has prompted a significant amount of research into *probabilistic logic programming* languages, both in the knowledge representation community itself, as well as in machine

learning, where such languages are developed for the purpose of *stochastic relational learning*.

Syntactically, such a language typically annotates a logic programming rule, or some part thereof, with a probability; the formal semantics of the language then somehow specifies a probability distribution—typically over a set of possible worlds—in terms of these individual probabilities. This is the way in which these probabilistic logic programming languages tend to be formally defined. However, such a formal definition still leaves one important question unanswered, namely that of how expressions in the langauge should be understood on the *informal* level, i.e., how would one explain their intuitive meaning to a non-logician?

For the two seperate components of logic programming and probability, this question has of course already been addressed at length. For instance, the informal meaning of logic programs—and in particular its negation-as-failure connective—has been explained among others in epistemic terms, referring to the beliefs of a rational agent (Gelfond and Lifschitz 1991), and in terms of the well-known mathematical concept of an inductive definition (Denecker 1998). The meaning of statements in probability calculus, on the other hand, has been explained among others in frequentist terms, e.g. (Venn 1866), and in terms of degrees of belief, e.g. (De Finetti 1937).

So far, research on probabilistic logic programming languages has not yet paid a great deal of attention to this issue of the informal meaning of expressions. It tends to be assumed that one already has sufficient intuitions about the meaning of logic programs and that the probabilities can simply be tacked on top of that. This paper presents an effort to develop a probabilistic logic programming langauge, whose informal semantics[1] is explained in full detail in a completely self-contained way. In general, the advantage of such an approach is that it gives more philosophical insight into the meaning of statements in the language, makes it easier to explain it to domain experts, and can help to provide a better modeling methodology for it.

One of the key tasks that such an effort needs to accomplish is to show convincingly that the formal semantics of the language indeed correctly captures the informal meaning that is attributed to its expressions, i.e., that these expressions indeed mean—formally—what we claim they—intuitively—mean. To ensure that this is done properly, we will adopt a constructive approach, where we first describe a particular kind of knowledge that we want to represent, then show how we can formalise the meaning of this knowledge in a way which is straightforward enough for its correctness to be intuitively obvious, and finally prove that the language we have thus defined is actually equivalent to a certain probabilistic logic programming construction.

The language that we develop will attempt to formalise *probabilistic causal laws*. The use of causal laws to compactly represent domains is commonplace in various

[1] The informal semantics of a language is commonly also referred to as its "intuitive reading". We prefer the term "informal semantics", however, because it stresses the close relation that there (should) exist(s) to the formal semantics. .

John and Mary are each holding a rock. With probability 0.5*, Mary will throw her rock at a window. This will break the window with probability* 0.8*. John will then also throw his rock at the window. He will hit it with probability* 0.6*.*

Fig. 1. The story of John and Mary.

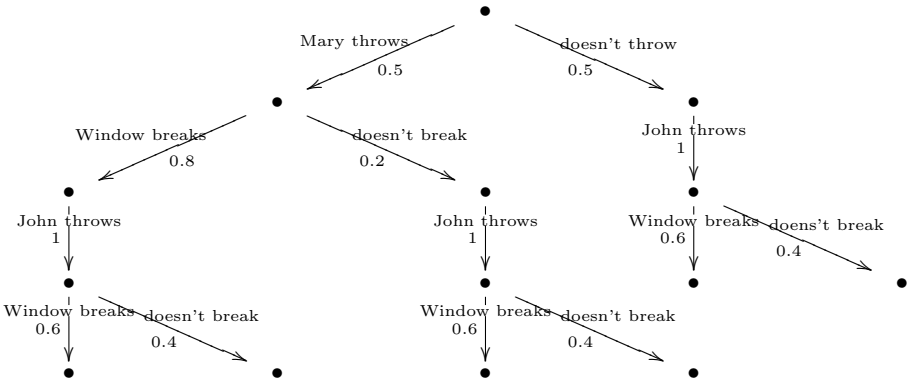


Fig. 2. Probability tree for the window breaking story.

action languages, related to logic programming, e.g. (Gelfond and Lifschitz 1993). Here, we will investigate a probabilistic variant of such laws. We will do this in the semantic context developed by Shafer (1996). In this work, Shafer presents his view on a number of fundamental causal and probabilistic concepts. His central hypothesis is that such concepts are best considered in an explicitly *dynamic* context: when speaking of probability or causality, we should do so, he says, in the context of a particular story about how the domain evolves, which he formalises by means of *probability trees*. As he himself puts it:

> A full understanding of probability and causality requires a language for talking about the structure of contingency—a language for talking about the step-by-step unfolding of events. This book develops such a language based on an old and simple yet general and flexible idea: the probability tree.

Figure 2 depicts a probability tree corresponding to the story shown in Figure 1. In natural language, we could say that such a tree paints the following picture. The domain starts out in an *initial state*. Then, some *event* happens, which causes the domain to transition to a new state. However, we do not know up front precisely which new state this is going to be, exactly; instead, the new state is chosen probabilistically from a set of alternatives. For instance, in the initial state of the tree in Figure 2, the event happens that Mary makes up her mind whether to throw, which leads to either a state in which she does or a state in which she does not. This step is then repeated—that is, in the new state, a different event happens, which leads to another new state, again chosen probabilistically from some set of alternatives—until finally this process arrives at a *final state*, in which no further events happen.

Throughout this paper, we will continue to talk about probability trees using the

language introduced above. In particular, we will carry on using the word *event* to refer to the occurrence that causes a transition from one state to the next. This use of the term differs from its standard use in probability theory, where it denotes a set of possible outcome of an experiment. Shafer introduces the terms *Humean event* (that which causes a transition between states) and *Demoivrian event* (a set of possible outcomes) to distinguish between these two different meanings of the word. Using the chain rule, the probability of following any particular branch in this tree can be computed as the product of the probabilities of the individual edges. For instance, the probability of the left-most branch of the tree in Figure 2 is $0.5 \cdot 0.8 \cdot 1 \cdot 0.6 = 0.24$. A Demoivrian event corresponds to a sets of branches of the tree. For instance, the Demoivrian event of the window being broken corresponds to the set of all branches in which it breaks, i.e., the first, second, third and fifth branch. The probability of such a Demoivrian event can be computed as the sum of the probabilities of all branches that belong to it, e.g., the probability of the window breaking is $0.24 + 0.16 + 0.06 + 0.3 = 0.76$. In the rest of this paper, we reserve the term "event" for Humean events (i.e., transitions between states) and will therefore omit the modifier.

This paper will develop a language for describing the causal laws according to which a probability tree unfolds. In other words, we will assume that each event in such a tree happens for a *reason*, i.e., that it is actually caused by some particular property of the state in which it happens. We then construct a language that allows to describe these reasons. In the extreme case, it might be that we can say nothing more than that each state of the tree is in itself the reason for the event that happens there; in this case, we obtain nothing more than an alternative representation for the tree itself. However, if the same event can happen in different parts of the tree, each time caused by the same property of the state in which it happens, we might end up with a significantly more compact representation.

In the story in Figure 1, we find four probabilistic causal laws:

- John throwing his rock causes the window to break with probability 0.6;
- Mary throwing her rock causes the window to break with probability 0.8;
- Mary decides to throw with probability 0.5 (this event is vacuously caused);
- John always throws (this event is also vacuously caused and it has only one possible outcome).

In the language that we will develop in the next section, we will write down such probabilistic causal laws in the format:

$$\textit{possible effects} \leftarrow \textit{cause}$$

where the *cause* can be omitted if the event is vacuously caused. In this syntax, the above probabilistic causal laws can be written down as:

$$(Break : 0.8) \leftarrow Throws(Mary). \tag{1}$$

$$(Break : 0.6) \leftarrow Throws(John). \tag{2}$$

$$(Throws(Mary) : 0.5). \tag{3}$$

$$Throws(John). \tag{4}$$

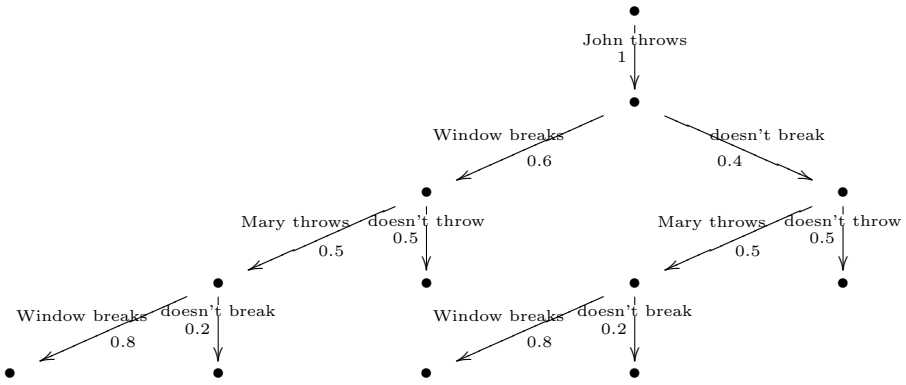


Fig. 3. Alternate probability tree for the window breaking story.

In this representation, John and Mary each get their own probabilistic causal law. This is necessary because they indeed throw differently, causing the window to break with different probability. However, we can also imagine an example in which both hit the window with the same probability. In this case, our language also allows a more compact representation, using a variable to range over the different persons that might throw:

$$\forall x \ (Break : 0.8) \leftarrow Throws(x).$$

The meaning of such a statement is as one would expect: each particular person that throws (i.e, each instantation of x for which $Throws(x)$ holds) hits the window with 0.8.

If we compare our four probabilistic causal laws to the probability tree in Figure 2, we see that this tree indeed obeys the causal laws, in the following sense:

- As we go down any branch of the tree, we find that all events which should happen according to our causal laws actually do so. The two unconditional causal laws (3) and (4) state that the events that Mary decides whether she will throw and that John decides that he will throw should always happen; and indeed, we find that in each branch of the tree, they do. In the left-most branch of the tree, for instance, the result of these two events is that both Mary and John decide to throw, so according to causal laws (1) and (2) the two events by which their respective throws break the window should also happen; and again, they indeed do.
- The events that happens according to the causal laws are also the only events that happen. For instance, in the right-most branch, Mary has decided not to throw, so the event of her rock breaking the window with probability 0.8 does not happen. Moreover, each of the events which should happen happens precisely once; it is not the case, for instance, that once Mary has decided to throw, the event of her rock breaking the window with probability 0.8 keeps on happening *ad infinitum*.

To define the semantics of our language, we will formalise this idea of a proba-

bility tree obeying a set of probabilistic causal laws. We will call such an obeying probability tree an *execution model* of the set of causal laws.

In general, a single set of causal laws might have many such execution models. Indeed, it is clear that a probability tree contains more information than the causal laws: events in the tree are totally ordered (for instance, in Figure 2, Mary decides to throw before John does), whereas the causal laws only provide a partial order on the events (for instance, the event of Mary's rock hitting the window can only happen after the event of Mary deciding to throw, since one causes the other; however, no order is imposed between e.g. the events of John throwing and Mary throwing). However, the additional information that is contained in a probability tree is actually irrelevant for the final outcome that will be reached. Let us consider, for instance, the alternative tree of Figure 3, in which John throws before Mary does. This tree also obeys our four causal laws, yet has a different structure than the tree in Figure 2. However, we see that the probability of the window eventually breaking is nevertheless precisely the same in this tree as it was in the other one, namely 0.76. Later in this paper, we will prove that all execution models of a set of causal laws always generate precisely the same probability distribution. In this sense, causal laws manage to capture the essence of a probability tree, while allowing irrelevant details (e.g. does John throw first or does Mary throw first?) to be ignored. This renders our representation quite succint.

Shafer's book also recognizes that the naive graphical representation of probability trees tends to grow unwieldy rather quickly. In the final chapter of his book, he therefore briefly examines a number of alternative, more compact representations for such trees, including Bayesian nets (Pearl 1988) and a representation based on Martin-Löf type theory (Martin-Löf 1982). In both these representations, new events are caused by the outcomes of some fixed set of previous events. The description of an event itself therefore already carries within it certain restrictions on the order in which events can happen. By constrast, in CP-logic events are not caused directly by previous events, but rather by properties of the state in which they happen. The fact that we do not represent any explicit *a priori* information about the order in which events happen makes our representation more flexible and allows probabilistic causal laws to easily be reused in different contexts. Let us suppose, for instance, that we know a probabilistic causal law that describes one particular way in which a certain disease can cause certain symptom. This law can then be reused, without change, regardless of what might cause the disease, which other causes there might be for the same symptom, or even whether there are still other ways in which the same disease might also cause the same symptom.

This paper is structured as follows. Section 2 briefly introduces some preliminary concepts from lattice theory and also logic programming. In Section 3, we formally define an initial, restricted version of CP-logic. In Section 4, we show how a certain kind of process can be modeled in this basic language, which also suggests a way of defining a more general version of CP-logic. This will be done in Section 5. Section 6 then discusses the resulting definitions in more detail. In Section 7, we investigate the precise relation between CP-logic and Bayesian networks. Section 8 relates CP-

logic to logic programming. Finally, Section 9 discusses some related work. Proof of the theorems presented in this paper will be given in Appendix Appendix A.

Part of the material in this paper was presented at conferences (Vennekens et al. 2004) and (Vennekens et al. 2006).

2 Preliminaries

This section recalls some well-known definitions and results from lattice theory and logic programming. To a large extent, this material is relevant only for the proofs of our theorems. It can safely be skipped on a first reading of this paper.

2.1 Some concepts from lattice theory

A binary relation $\leq$ on a set L is a *partial order* if it is reflexive, transitive and anti-symmetric. A partially ordered set $\langle L, \leq \rangle$ is a *lattice* if every pair (x, y) of elements of L has a unique least upper bound and greatest lower bound. Such a lattice $\langle L, \leq \rangle$ is *complete* if every non-empty subset $S \subseteq L$ has a least upper bound and greatest lower bound. A complete lattice has a least element $\bot$ and a greatest element $\top$. An operator $O : L \to L$ is *monotone* if for every $x \leq y$, $O(x) \leq O(y)$. An element $x \in L$ is a *prefixpoint* of O if $x \geq O(x)$, a *fixpoint* if $x = O(x)$ and a *postfixpoint* if $x \leq O(x)$. If O is a monotone operator on a complete lattice, then for every postfixpoint y, there exists a least element in the set of all prefixpoints x of O for which $x \geq y$. This least prefixpoint greater than y of O is also the least fixpoint greater than y of O. Moreover, it can be constructed by successively applying O to y, i.e., as the limit of the sequence $(y, O(y), O(O(y)), \ldots)$. In particular, because $\bot$ is a trivial postfixpoint, O has a least prefixpoint which is equal to its least fixpoint and which can be constructed by successive application of O to $\bot$.

2.2 Some concepts from logic programming

We assume familiarity with classical logic. A *Herbrand* interpretation for a vocabulary Σ is an interpretation, which has as its domain the set $HU(\Sigma)$ of all ground terms that can be constructed from Σ and which interprets each constants as itself and each function symbol f/n as the function mapping a tuple $(t_1, \ldots, t_n)$ to $f(t_1, \ldots, t_n)$. We can identify a Herbrand interpretation with a set of ground atoms. A *partial* Herbrand interpretation is a function ν from the set $HB(\Sigma)$ of all ground atoms, also called the *Herbrand base*, to the set of truth values $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. A (total) Herbrand interpretation corresponds to a partial Herbrand interpretation that does not include $\mathbf{u}$ in its range. On the set of truth values, one defines the *precision order* :

$$\mathbf{u} \leq_p \mathbf{f} \text{ and } \mathbf{u} \leq_p \mathbf{t}$$

and the *truth order*:

$$\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}.$$

These orders can be pointwise extended to partial Herbrand interpretations. Each totally ordered set S of partial Herbrand interpretations has a $\leq_p$-least upperbound denoted $lub_{\leq_p}(S)$. The three-valued truth function φ^ν for sentences φ and partial Herbrand interpretations ν is defined by induction:

- $p^\nu = \nu(p)$, for $p \in HB(\Sigma)$;
- $(\psi \wedge \phi)^\nu = Min_{\leq_t}(\psi^\nu, \phi^\nu)$;
- $(\forall x\ \phi(x))^\nu = Min_{\leq_t}(\{\phi(t)^\nu | t \in HU(\Sigma)\})$.
- $(\neg\varphi)^\nu = (\varphi^\nu)^{-1}$ where $\mathbf{f}^{-1} = \mathbf{t}, \mathbf{t}^{-1} = \mathbf{f}, \mathbf{u}^{-1} = \mathbf{u}$.

A crucial monotonicity property of three-valued logic is that $\nu \leq_p \nu'$ implies $\varphi^\nu \leq_p \varphi^{\nu'}$.

The well-founded semantics of logic programs was originally defined in (Van Gelder et al. 1991). We present an equivalent definition that was developed in (Denecker and Vennekens 2007). Formally, a logic program P is a set of rules of the form $p \leftarrow \phi$, where p is a ground atom and ϕ is a first-order sentence.

Definition 1 (*well-founded induction*)
We define a *well-founded induction* of P as a sequence of partial Herbrand interpretations $(\nu^\alpha)_{0 \leq \alpha \leq \beta}$ satisfying the following conditions:

- $\nu^0 = \bot_{\leq_p}$, the mapping of all atoms to $\mathbf{u}$;
- $\nu^\lambda = lub_{\leq_p}(\{\nu^\beta | \beta < \lambda\})$, for each limit ordinal λ;
- $\nu^{\alpha+1}$ relates to ν^α in one of the following ways:
  - $\nu^{\alpha+1} = \nu^\alpha[p : \mathbf{t}]$ such that for some rule $p \leftarrow \varphi$ in P, $\varphi^{\nu^\alpha} = \mathbf{t}$;
  - $\nu^{\alpha+1} = \nu^\alpha[U : \mathbf{f}]$ where U is an *unfounded set*, i.e., a set of ground atoms such that for each p in U, $\nu^\alpha(p) = \mathbf{u}$ and for each rule $p \leftarrow \varphi$ in P, $\varphi^{\nu^{\alpha+1}} = \mathbf{f}$.

A well-founded induction is a sequence of increasing precision. We call a well-founded induction $(\nu^\alpha)_{\alpha \leq \beta}$ *terminal* if it cannot be extended with a strictly more precise interpretation. Each well-founded induction whose limit is a total interpretation is terminal. We now define the *well-founded model* of P as the limit of any such terminal well-founded induction. As the following result shows, this definition coincides with the standard one.

Theorem 1
(Denecker and Vennekens 2007) Each terminal well-found induction of P converges to the well-founded model of P, as it was defined in (Van Gelder et al. 1991).

In certain logic programming variants, such as abductive logic programs (Kakas et al. 1992) and ID-logic (Denecker and Ternovska 2007), a distinction is made between predicates that are *defined* by the program and predicates that are left *open*. The set of defined predicates must contain at least those predicates that appear in the heads of rules of the program. This distinction is similar to that between endogenous and exogenous random variables, which is common in probabilistic modeling. It is straightforward to generalize the well-founded semantics to this case. Given an interpretation O of the open predicates, we define a *well-founded*

induction of P in O by the same inductive definition as for ordinary well-founded inductions, only we now have as a base case that ν^0 should be the least precise partial Herbrand interpretation *that extends* O. It is easy to see that each ν^i in such a well-founded induction in O in fact extends O and also that if there are no open predicates, this definition simply coincides with the original one. The *well-founded model of P in O* is then the limit of any terminal well-founded induction of P in O.

3 A logic of probabilistic causal laws

Our goal in this section is to define a language for representing probabilistic causal laws. Before going into the mathematical details, we first outline the general picture.

To represent knowledge in a logical language, the first thing that is needed is a suitable vocabulary. Usually in logical modeling, this vocabulary is assumed to be such that each possible state of the domain of discourse corresponds to an interpretation for it. In Shafer's probability trees, possible states of the domain are represented by nodes of the tree. To link these two formal settings, our semantics will therefore consider probability trees in which each node corresponds to an interpretation for a given vocabulary.

As we introduced the concept in Section 1, a probabilistic causal law states the *cause* and *possible effects* of a particular event or class of events. The cause specifies in which nodes of the tree the event might happen, i.e, it is some property of the domain of discourse such that the event can happen in precisely those states of the domain in which this property holds. The natural thing, therefore, is to represent such a cause by a first-order formula ϕ, whose meaning is that the event might happen in those states s of a probability tree for which the associated interpretation $\mathcal{I}(s)$ is such that $\mathcal{I}(s) \models \phi$.

Each event that happens makes a transition from a node s of a probability tree to one of the children s' of s. The description of the effects of such an event should specify how it will affect the state of the domain, i.e., what the interpretations $\mathcal{I}(s')$ associated to the children s' of s should be. There are many conceivable ways of representing such knowledge, but in this paper we stick to a very simply one: we assume that each possible effect of an event corresponds to a single ground atom $P(\mathbf{t})$ of our vocubalary, such that the interpretation $\mathcal{I}(s')$ corresponding to the new state s' is indentical to the interpretation $\mathcal{I}(s)$, apart from the fact that $P(\mathbf{t})$ is now true. We choose this simple representation for two reasons. First, the aim of our exercise is to come up with a semantics that formalises probabilistic causal laws in a way that clearly coincides with our intuitions about the meaning of such laws. Keeping the representation of effects simple helps to achieve the desired clarity. Second, we are not just interested in this language for its own sake, but also because we want to use it to explain the meaning of certain probabilistic logic programming statements. Our simple representation of effects will also serve to elucidate this link to logic programming.

3.1 Syntax

In this section, we formally define the language of CP-logic. Let us fix a finite relational vocabulary, consisting of a set of predicate symbols and a set of constants. We assume that the predicates of our vocabulary are split into a set of *endogenous* predicates and a set of *exogenous* ones. The idea behind this distinction is of course that the endogenous predicates should describe things that are internal to the causal process being modeled, while the exogenous predicates describe things external to it.

A *causal probabilistic law*, or *CP-law* for short, is a statement of the form:

$$\forall \mathbf{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \phi, \tag{5}$$

where the α_i are non-zero probabilities with $\sum \alpha_i \leq 1$, ϕ is a first-order formula and the A_i are atoms, such that the universally quantified tuple of variables $\mathbf{x}$ contains all free variables in ϕ and the A_i. Moreover, the predicate symbol of each of the atoms A_i should be an endogenous predicate.

Such a CP-law is read as:

"For each $\mathbf{x}$, ϕ causes an event whose effect is that at most one of the A_i becomes true; for each i, the probability of A_i being the effect of this event is α_i."

If the causal law has a deterministic effect, i.e., it causes some atom A with probability 1, we also write $A \leftarrow \phi$ instead of $(A : 1) \leftarrow \phi$. We allow the precondition ϕ to be absent, meaning that the event is vacuously caused. In this case, the causal law is called *unconditional* and we omit the '$\leftarrow$'-symbol as well. If the tuple $\mathbf{x}$ of variables is empty, we call the causal law *ground*. We remark that the precondition ϕ of such a ground causal law may still contain variables, as long as they are all bound by some quantifier in ϕ.

A *CP-theory* is a finite multiset[2] of CP-laws. For now, we will restrict attention to CP-theories in which all formulas ϕ are positive, i.e., they do not contain negation. Afterwards, Section 5 will examine how negation can be added.

Example 1
In about 25% of the cases, *syphilis* causes a neuropsychiatric disorder called *general paresis*, and in fact, syphilis is the only cause for paresis. This can be modeled as follows:

$$(Paresis : 0.25) \leftarrow Syphilis. \tag{6}$$

where $Syphilis$ is an exogeneous predicate. This example illustrates the difference between causation and material implication. Indeed, because syphilis is the only cause for paresis, observing that a patient has paresis implies that he must also have syphilis, i.e., the material implication $Paresis \supset Syphilis$ holds. So, in this example, the directions of causation and material implication are precisely opposite.

[2] Example 3 explains why we consider multisets instead of sets.

Example 2
Our running example for this section will also be a medical example. Pneumonia might cause angina with probability 0.2. Vice versa, angina might cause pneumonia with probability 0.3. A bacterial infection can cause either pneumonia (with probability 0.4) or angina (with probability 0.1). We consider bacterial infection as exogeneous.

$$(Angina : 0.2) \leftarrow Pneumonia. \tag{7}$$

$$(Pneumonia : 0.3) \leftarrow Angina. \tag{8}$$

$$(Pneumonia : 0.4) \vee (Angina : 0.1) \leftarrow Infection. \tag{9}$$

Example 3
A CP-theory is a *multiset* of CP-laws, which means that it may contain several instances of the same event. To illustrate this, consider a variant of the above problem in which the patient comes into contact with two different sources of infection, each of which might cause him to become infected with a probability of 0.1. To model this, we can add the following multiset of two unconditional events to the theory of Example 2:

$$(Infection : 0.1). \tag{10}$$

$$(Infection : 0.1). \tag{11}$$

We now define some notation to refer to different components of a ground CP-law. The head $head(r)$ of a rule r of form (5) is the set of all pairs (A_i, α_i) appearing in the description of the effects of the event; the body of r, $body(r)$, is its precondition ϕ. By $head_{At}(r)$ we denote the set of all atoms A_i for which there exists an α_i such that $(A_i, \alpha_i) \in head(r)$. Similarly, by $body_{At}(r)$ we will denote the set of all atoms A which "appear"[3] in $body(r)$. For the above Example 2, if r is the CP-law (9), then $head(r) = \{(Pneumonia, 0.4), (Angina, 0.1)\}$, $head_{At} = \{Pneumonia, Angina\}$, $body(r) = Infection$ and $body_{At}(r) = \{Infection\}$.

We will call a CP-law $E \leftarrow \phi$ a *rule* if we want to emphasize that we are referring to a syntactical construct.

3.2 Semantics

This section defines the formal semantics of CP-logic. We will restrict attention to Herbrand interpretations, i.e., we consider only interpretations whose domain is the set of constants of the theory and which interpret each constant as itself. This

[3] More formally, we use $body_{At}(r)$ to denote $At(body(r))$, where At is the mapping from sentences to sets of ground atoms, that is inductively defined by:

- For $Q(\mathbf{t})$ a ground atom, $At(Q(\mathbf{t})) = \{Q(\mathbf{t})\}$;
- For $\phi \circ \psi$, with $\circ$ either $\vee$ or $\wedge$, $At(\phi \circ \psi)$ is defined as $At(\phi) \cup At(\psi)$;
- For $\neg\phi$, $At(\neg\phi) = At(\phi)$;
- For $\Theta x\ \phi$, with Θ either $\forall$ or $\exists$, $At(\Theta x\ \phi) = \cup_{t \in H_U(\Sigma)} At(\phi[x/t])$, where $H_U(\Sigma)$ denotes the Herbrand universe for the vocabulary Σ.

restriction is made for two reasons: it simplifies the presentation, and it is also what is usually done in (probabilistic) logic programming. However, it is easy to extend all our definitions and results to arbitrary domains.

We view a non-ground CP-law $\forall \mathbf{x}\ r$ as an abbreviation for the set of all ground CP-laws $r[\mathbf{x}/\mathbf{t}]$ that result from replacing the variables $\mathbf{x}$ by a tuple $\mathbf{t}$ of ground terms in vocabulary Σ. For instance, if we wanted to consider multiple people in Example 2, we might include constants $\{John, Mary\}$ in our vocabulary Σ and write the non-ground rule

$$\forall x\ (Angina(x) : 0.2) \leftarrow Pneumonia(x),$$

to abbreviate the two CP-laws

$$(Angina(John) : 0.2) \leftarrow Pneumonia(John).$$
$$(Angina(Mary) : 0.2) \leftarrow Pneumonia(Mary).$$

Because CP-theories are finite, the use of such abbreviations only makes sense in the context of a finite domain, i.e., when the vocabulary does not generate an infinite number of terms. By restricting attention to finite relational vocabularies, we ensure that this is the case.

In our formal treatment of CP-logic, we will never consider non-ground rules, but always assume that these have already been expanded into a finite set of ground CP-laws. When using such non-ground rules in examples, we will implicitly assume that predicates and constants have been appropriately typed, in such a way as to avoid instantiations that are obviously not intended. We will also allow ourselves to use arithmetic function symbols, such as +/2 and -/2, and assume that the grounding replaces terms made from these symbols by numerical constants in the appropriate way.

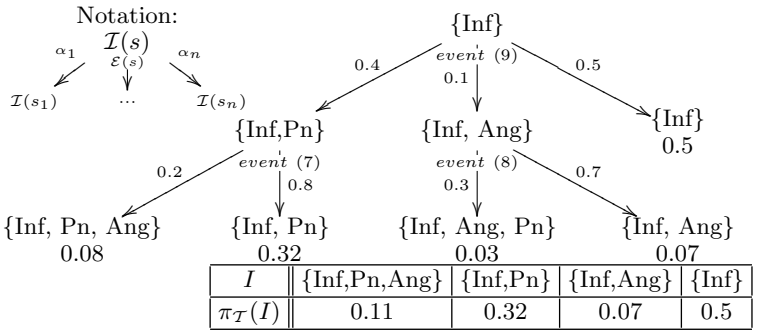
As already explained, our basic semantical object will be that of a probability tree in which the nodes correspond to interpretations.

Definition 2 (*probabilistic* Σ*-process*)
Let Σ be a vocabulary. A *probabilistic* Σ*-process* $\mathcal{T}$ is a pair $\langle T; \mathcal{I} \rangle$, where:

- T is a tree structure, in which each edge is labeled with a probability, such that for every non-leaf node s, the probabilities of all edges leaving s sum up to precisely 1;
- $\mathcal{I}$ is a mapping from nodes of T to Herbrand interpretations for Σ.

In a probability tree, we can associate to each node s the probability $\mathcal{P}(s)$ of a random walk in the tree, starting from its root, passing through s. Indeed, for the root $\bot$ of the tree, $\mathcal{P}(\bot) = 1$ and for each other node s, $\mathcal{P}(s) = \prod_i \alpha_i$ where the α_i are all the probabilities associated to edges on the path from the $\bot$ to s. Essentially, the mapping $\mathcal{P}$ contains all the information that is present in the labeling of the edges and vice versa. To ease notation, we will sometimes take the liberty of identifying a probabilistic Σ-process $\langle T; \mathcal{I} \rangle$ with the triple $\langle T; \mathcal{I}; \mathcal{P} \rangle$ and ignoring the labels on the edges of T.

Each probabilistic Σ-process now induces an obvious probability distribution over the states in which the domain described by Σ might end up.



I	{Inf,Pn,Ang}	{Inf,Pn}	{Inf,Ang}	{Inf}
$\pi_{\mathcal{T}}(I)$	0.11	0.32	0.07	0.5

Fig. 4. A process $\mathcal{T}$ for Example 2 and its distribution $\pi_{\mathcal{T}}$.

Definition 3 ($\pi_{\mathcal{T}}$)
Let Σ be a vocabulary and $\mathcal{T} = \langle T; \mathcal{I}; \mathcal{P} \rangle$ a probabilistic Σ-process. By $\pi_{\mathcal{T}}$ we denote the probability distribution that assigns to each Herbrand interpretation I of Σ the probability $\sum_{s \in L_{\mathcal{T}}(I)} \mathcal{P}(s)$, where $L_{\mathcal{T}}(I)$ is the set of all leaves s of T for which $\mathcal{I}(s) = I$.

Like any probability distribution over interpretations, such a $\pi_{\mathcal{T}}$ also defines a set of *possible worlds*, namely that consisting of all I for which $\pi_{\mathcal{T}}(I) > 0$. If all the probabilities $\mathcal{P}(s)$ are non-zero, then this is simply the set of all $\mathcal{I}(l)$ for which l is a leaf of $\mathcal{T}$.

We now want to relate the transitions in such a probabilistic Σ-process to the events described by a CP-theory.

Definition 4 (*rules firing*)
Let Σ be a vocabulary, C a CP-theory in this vocabulary and $\mathcal{T}$ a probabilistic Σ-process. Let $r \in C$ be a CP-law of the form:

$$(A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \phi.$$

We say that r *fires* in a node s of $\mathcal{T}$ if s has $n+1$ children $s_1, \ldots, s_{n+1}$, such that:

- For all $1 \leq i \leq n$, $\mathcal{I}(s_i) = \mathcal{I}(s) \cup \{A_i\}$ and the probability of edge (s, s_i) is α_i;
- For s_{n+1}, $\mathcal{I}(s_{n+1}) = \mathcal{I}(s)$ and the probability of the edge (s, s_{n+1}) is $1 - \sum_i \alpha_i$.

For simplicity, we will omit edges labeled with a probability of zero; this does not affect any of the following material.

This definition now allows us to link the transitions in a probabilistic Σ-process $\mathcal{T}$ to the events of a CP-theory C. Formally, we will consider a mapping $\mathcal{E}$ from each non-leaf node s of $\mathcal{T}$ to an associated CP-law $r \in C$. Because the same ground CP-law should fire at most once in each branch, the following definition will also consider, for a node s, the set of all CP-laws that have not yet fired in s, i.e., the set of all $r \in C$ for which there does not exist an ancestor s' of s such that $\mathcal{E}(s') = r$. We will denote this set as $\mathcal{R}_{\mathcal{E}}(s)$.

Definition 5 (*execution model–positive case*)

Let C be a positive CP-theory and X an interpretation of the exogenous predicates. A probabilistic Σ-process $\mathcal{T} = \langle T; \mathcal{I} \rangle$ is an *execution model* of C in context X, written $\mathcal{T} \models_X C$, iff there exists a mapping $\mathcal{E}$ from the non-leaf nodes of $\mathcal{T}$ to C, such that:

- For the root $\bot$ of $\mathcal{T}$, $\mathcal{I}(\bot) = X$;
- In each non-leaf node s, a CP-law $\mathcal{E}(s) \in \mathcal{R}_{\mathcal{E}}(s)$ fires, such that its precondition is satisfied in s, i.e., $\mathcal{I}(s) \models \mathit{body}(\mathcal{E}(s))$;
- For each leaf l of $\mathcal{T}$, there are no CP-laws $r \in \mathcal{R}_{\mathcal{E}}(l)$ for which $\mathcal{I}(l) \models \mathit{body}(r)$.

If there are no exogenous predicates, we simply write $\mathcal{T} \models C$.

Example 2 has one execution model for every specific context X; the process for $X = \{Infected\}$ is depicted in Figure 4. As we showed with the window breaking example in Section 1, there also exist thoeries which allow multiple execution models for a given context. However, all of these execution models must then generate the same probability distribution over their final states.

Theorem 2 (*Uniqueness—positive case*)
Let C be a positive CP-theory. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are both execution models of C, then $\pi_{\mathcal{T}_1} = \pi_{\mathcal{T}_2}$.

Proof
Proof of this theorem can be found in Section A.2. □

This theorem shows that knowing all the probabilistic causal laws of a domain gives enough information to predict a single probability distribution over the final states that this domain might reach. This result is important for two reasons.

First, it suggests an appealing explanation for why causality is such a useful and important concept: causal information tells you just enough about the behaviour of a probabilistic process to be able to predict its final outcome in every possible context, while allowing irrelevant details to be ignored. As such, it offers a compact and robust representation of the class of probability distributions that can result from such a process. Second, in our construction of CP-logic, we have started from Shafer's dynamic analysis of causality, using the probability tree as a basic semantic object. In this respect, our approach differs from that of causal Bayesian networks (Pearl 2000), in which causal information is viewed more statically, with probability distributions as basic semantical objects. The above theorem relates these two views, because it allows us to not only view a CP-theory as describing a class of processes, but also as defining a unique probability distribution.

Definition 6 (π_C^X)
Let C be a CP-theory and X an interpretation for the exogenous predicates of C. By π_C^X, we denote the unique probability distribution $\pi_{\mathcal{T}}$, for which $\mathcal{T} \models_X C$. If there are no exogenous predicates, we simply write π_C.

A CP-theory can be viewed as mapping each interpretation for the exogenous predicates to a probability distribution over interpretations of the endogenous predicates or, to put it another way, as a conditional distribution over interpretations of the endogenous predicates, given an interpretation for the exogenous predicates.

Definition 7 (*models of a CP-theory*)
Let C be a CP-theory and π a probability distribution over interpretations of all the predicates of C. π is a *model* of C, denoted $\pi \models C$ iff for each interpretation X of the exogenous predicates with $\pi(X) > 0$ and each interpretation J of the endogenous predicates, $\pi(J \mid X) = \pi_C^X(J)$.

If a CP-theory C has no exogenous predicates, then there is a unique π for which $\pi \models C$ and this is, of course, simply the distribution π_C.

Having defined this formal semantics for CP-logic, it is natural to ask how the causal interpretation that we have informally attached to expressions in our language is reflected in it. We see that our semantics essentially consists of the following two causal principles:

- The principle of *universal causation* states that all changes to the state of the domain must be triggered by a causal law whose precondition is satisfied.
- The principle of *sufficient causation* states that if the precondition to a causal law is satisfied, then the event that it triggers must eventually happen.

Together with our decision to use Shafer's probability trees as our basic semantical objects and the particular representation that we have chosen for probabilistic causal laws, these two principles essentially determine our logic completely—at least, in the positive case. In the following sections, we turn our attention to the question of how to extend our definitions to the case where negation can appear in the precondition of a CP-law. However, this requires us to first discuss in more detail a particular modeling methodology for CP-logic.

4 Modeling more complex processes in CP-logic

In the formal semantics of CP-logic, the interpretations $\mathcal{I}(s)$ associated to nodes s of the probability trees play an important role. Indeed, if we forget for a moment the restriction that each causal law can fire at most once in each branch, then the interpretation $\mathcal{I}(s)$ completely determines which of the causal laws can fire in s. In our account of CP-logic so far, we have suggested that the logical vocabulary Σ of a CP-theory be chosen in such a way that possible states of the domain of discourse correspond to (Herbrand) interpretations for Σ. However, this assumption restricts the kind of causal laws that can be represented in at least two different, but related, ways. First, it means that we can only describe events that are caused by some property of the *current* state s. In particular, it is not possible to say that an event is caused by something which happened *previously*, but no longer has any visible effect on the current state. Second, since the interpretations $\mathcal{I}(s)$ grow monotonically throughout a branch, i.e., no atoms are ever removed from such an interpretation, it also means that we actually cannot even describe events whose effects manifest themselves in some state, but then disappear again in a future state. The following example illustrates these limitations of the view that an interpretation $\mathcal{I}(s)$ represents precisely the state of the domain at node s.

Example 4
In 10% of the cases, pneumonia causes permanent lung damage, which persists after the pneumonia itself has disappeared. Let us also assume that the probability of getting pneumonia is 0.3. One attempt to model this is as follows:

$$(LungDamage : 0.1) \leftarrow Pneumonia. \tag{12}$$

$$(Pneumonia : 0.3). \tag{13}$$

The problem with this theory is that, under the natural interpretation of the predicates, it violates the assumptions made by CP-logic: after pneumonia has been caused and has in turn lead to permanent lung damage, it might go away again. As such, it is no surprise that, according to the formal semantics of this theory, the probability of a patient having permanent lung damage and no pneumonia is zero, while in reality, this situation is perfectly possible.

There is however a simple solution to this problem—at least, if we are prepared to refine our informal interpretation of the atom *Pneumonia*. Instead of interpreting this atom as representing the real-world property that "the patient has pneumonia", we can also interpret it as representing the property that "at some point in time, the patient has had pneumonia". It is obvious that this now *is* a property that, once initiated, will forever persist. The CP-law (12) now reads as: "if the patient has, at some point, had pneumonia, then this causes him to have lung damage with probability 0.1." According to this reading, it is now only the case that it is impossible for a patient to have lung damage if he has not *at some point in time* had pneumonia, which is of course a conclusion that should follow from our problem statement.

To fix this example, we had to subtly change the correspondence between the states of the formal execution model and the states of the real world: whereas previously, each of our formal states precisely matched one state of the real world, it is now the case that a formal state actually represents both the state of the world at some particular time *and* also certain information about the history of the world up to that time. Taking this idea further actually allows us to describe processes in considerable temporal detail, as the following example illustrates.

Example 5
A patient is admitted to hospital with pneumonia and stays there for a number of days. Each day, the pneumonia might cause him to suffer chest pain on that particular day with probability 0.6. With probability 0.8, a patient who has pneumonia on one day still has pneumonia the next day.

On the one hand, this example describes a progression through a sequence of days. On the other hand, it also describes events that takes place entirely during one particular day. In general, a process of this kind will look something like Figure 5: the global structure of the process is a succession between different time points and, at each particular time point, a local process might take place.

The question now is how to model such a succession of states in CP-logic. A first important observation is that we now need to distinguish between the values of

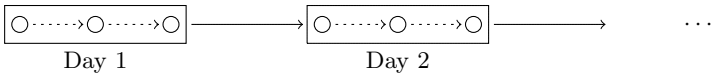


Fig. 5. A global process as a sequence of local processes.

properties at different time points, i.e., we can no longer represent every relevant property by a single ground atom, but instead we need a ground atom for every pair of such a property and a time point. Typically, one would construct a vocabulary by adding time as an argument to predicates, as is done in, e.g., the event calculus or situation calculus. For instance, to describe Example 5, we could construct a vocabulary which has the following ground atoms:

- Referring to day 1: $\{Pneumonia(1), Chestpain(1)\}$;
- Referring to day 2: $\{Pneumonia(2), Chestpain(2)\}$;
- ...
- Referring to day n: $\{Pneumonia(n), Chestpain(n)\}$;

Of course, it might be equally possible to use some other representation, such as $Pneumonia(SecondDay)$ or $Pneumonia2$ instead of $Pneumonia(2)$. With the above vocabulary, we can now model Example 5. We assume a fixed range $1..n$ of days, to ensure finiteness of the grounded theory.

$$Pneumonia(1). \tag{14}$$

$$\forall d\ (Pneumonia(d+1) : 0.8) \leftarrow Pneumonia(d). \tag{15}$$

$$\forall d\ (Chestpain(d) : 0.6) \leftarrow Pneumonia(d). \tag{16}$$

Here, the CP-laws described by (15) are of the kind that propagate from one time point to a later time point, whereas (16) describes a class of "instantaneous" events, taking place entirely inside of a single time point. Of course, whether a particular event is instantaneous depends greatly on which unit of time is being used: one can imagine that it makes a difference whether we measure time in seconds or in days.

According to the informal description of Example 5, the intended model is the process shown in Figure 6. It can easily be seen that this is indeed an execution model of the above CP-theory. We remark that this theory also has other execution models, which do not respect the proper ordering of time points, such as, e.g., the process in which all events caused by (15) happen before those caused by (16). However, since these "wrong" processes all generate the same probability distribution as the intended process anyway, this is harmless.

We also observe that, again, the correspondence between the states of the execution model and the states of the real world is less direct than it was in the examples of Section 3.2. Indeed, now, a state of an execution model contains a trace of the entire evolution of the real world until a certain point in time. As such, a leaf of the execution model now represents a complete history of the world, whereas in the examples of Section 3.2, it only represented the final state of the process.

Let us now make the above discussion more formal. We assume that, when constructing the vocabulary Σ, we had in mind some function λ from the Herbrand base of Σ to an interval $[0..n] \subseteq \mathbb{N}$, such that, in our desired interpretation of this

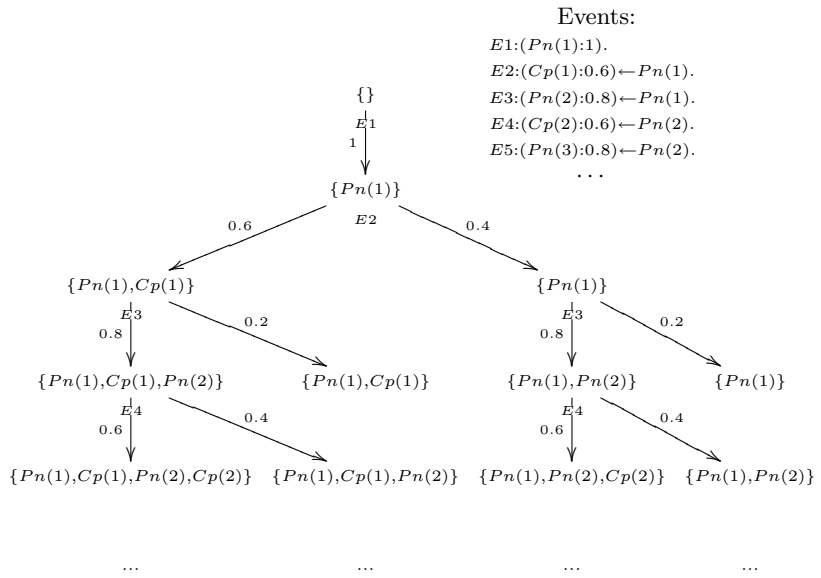


Fig. 6. Initial segment of the intended model of Example 5.

vocabulary, each atom p refers to the state of some property at time point $\lambda(p)$. We call such a function a *timing* and $\lambda(p)$ the *time* of atom p. In the typical case of predicates containing an explicit temporal argument, such a timing would simply map atoms onto this argument; for instance, in the case of the above example, we had in mind the following timing λ:

- For each ground atom $Pneumonia(i)$, $\lambda(Pneumonia(i)) = i$;
- For each ground atom $Chestpain(i)$, $\lambda(Chestpain(i)) = i$.

If we now look again at the CP-laws we wrote for this example, we observe that, whenever there is an atom in the head of a CP-law r that refers to the truth of some property at time i and an atom in the body of r that refers to the truth of some property at time j, it is the case that $i \geq j$. This is of course not a coincidence. Indeed, because, in the real world, causes precede effects, it should be impossible that the cause-effect propagation described by a CP-law goes backwards in time. Note that it is also possible that $i = j$; in this case, the CP-law is instantaneous w.r.t. the granularity of time that is being used, i.e., it describes one of those events (such as (15) in Example 5) that takes place entirely within a single time point. Another, perhaps more illustrative, example of an instantaneous CP-law is the statement that an increase in the current flowing through a resistor causes an increase in the voltage drop across it. Here, the increased current *conceptually* precedes the increased voltage drop, but we would never expect to actually observe a temporal delay.

Definition 8 (*respecting a timing*)
Let Σ be a vocabulary. A CP-theory C respects a timing λ iff, for every $r \in C$, if $h \in head_{At}(r)$ and $b \in body_{At}(r)$, then $\lambda(h) \geq \lambda(b)$.

Such a timing λ also contains information about when events might happen. To be more concrete, if a CP-law r fires at time point i, then we would expect i to lie somewhere between the maximum of all $\lambda(b)$ for which $b \in body_{At}(r)$, and the minimum of all $\lambda(h)$ for which $h \in head_{At}(r)$. For a rule r, we write $t_\lambda(r)$ to denote this interval, i.e.,

$$t_\lambda(r) = [\max_{b \in body_{At}(r)} \lambda(b), \min_{h \in head_{At}(r)} \lambda(h)].$$

Now, if we are constructing a CP-theory with a particular timing λ in mind, then the process we are trying to model should be such that every CP-law r that actually fires does so at some time point $\kappa(r) \in t_\lambda(r)$. We will call such a mapping κ from rule $r \in C$ to time points $\kappa(r) \in t_\lambda(r)$ an *event-timing* of λ. We remark that if a CP-law r is instantaneous, then the interval $t_\lambda(r)$ will consist of a single time point and it is indeed clearly at this time point that the CP-law should fire.

A timing λ therefore imposes the following constraint on which processes can be considered reasonable.

Definition 9 (*following a timing*)
Let Σ be a vocabulary with timing λ and let C be a CP-theory that respects λ. A probabilistic Σ-process $\mathcal{T}$ *follows* λ if there exists an event-timing κ of λ such that the CP-laws of $\mathcal{T}$ fire in the order imposed by κ, i.e., if for all successors s' of a node s, $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$.

It can now be shown that for any timing λ and any CP-theory C respecting λ, C will have an execution model that follows λ.

Theorem 3
Let C be a CP-theory respecting a timing λ. There exists an execution model $\mathcal{T}$ of C, such that $\mathcal{T}$ follows λ.

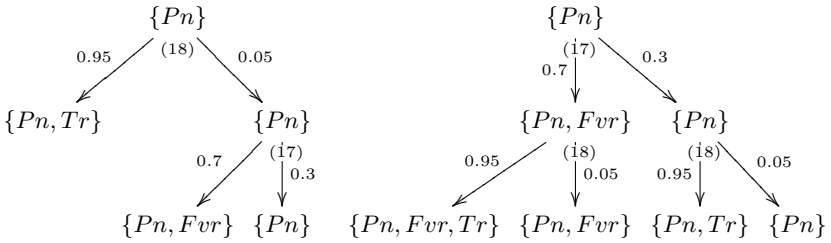
Proof
Proof of this theorem can be found in Section A.3. □

This result shows that if we construct a CP-theory C with a particular timing in mind, then C will have an execution model in which the events happen in precisely the order dictated by this timing. Therefore, the modeling methodology that we have suggested in this section is indeed valid. In the case of Example 5, the process shown in Figure 6 is an execution model that follows the timing λ specified above.

In the sequel, we will refer to CP-theories, for whose vocabulary we have some intended timing in mind, as *temporal CP-theories*; other CP-theories will be called *atemporal*.

5 CP-logic with negation

So far, we have only allowed positive formulas as preconditions of CP-laws. In this section, we examine whether it is possible to relax this requirement. We first look at a small example.



(a) Sequence (18)–(17). (b) Sequence (17)–(18).

Fig. 7. Two processes for Example 6.

Example 6
Having pneumonia causes a patient to receive treatment with probability 0.95. Untreated pneumonia causes fever with probability 0.7.

$$(Fever : 0.7) \leftarrow Pneumonia \wedge \neg Treatment. \tag{17}$$

$$(Treatment : 0.95) \leftarrow Pneumonia. \tag{18}$$

Figure 7 shows two processes for this example that satisfy all the requirements that we previously imposed for positive theories. It is obvious, however, that in this case the final outcome is affected by the order in which events occur. So, simply including negation in this naive way would give rise to ambiguities, causing our desirable uniqueness property (Theorem 2) to be lost.

Giving up the uniqueness property would have grave consequences for the logic and its practical use. One radical solution to the problem might be to force the user to not only specify causal probabilistic events, but also information about the order in which these events can happen. However, such information is difficult to obtain and represent; moreover, in many cases, it would just be useless overhead—indeed, as we have already seen, one of the most interesting features of CP-logic without negation is precisely the fact that we can obtain a complete probability distribution *without* requiring such information. The solution that we will adopt instead is to restrict the class of processes associated to a CP-theory in such a way that the uniqueness property is preserved, i.e., all processes from this restricted class generate the same distribution over the final states.

To introduce the additional constraint that will be imposed on execution models, let us take a closer look at the above example. We observe that, in process 7(b), event (17) is caused at a moment when its precondition is not yet in its *final* state. In particular, when (17) happens in the initial state, its precondtion $\neg Treatment$ holds, but later on, for instance in the leftmost branch, event (18) causes $Treatment$, thereby falsifying this precondition. So, in the final state of this branch, we see that $Fever$ holds, while the precondition of the CP-law that caused it no longer holds.

In light of this discussion, we can now explain the additional assumption that CP-logic makes about the causal processes. This assumption, called the *temporal precedence assumption*, is that a CP-law r will not fire until its precondition is in its final state. More precisely, it cannot fire *until the part of the process that*

determines whether its precondition holds is fully finished. For Example 6, it is clear that only the process in Figure 7(a) satisfies this assumption, and so, in this case, the ambiguity has been resolved.

We stress here that temporal precedence is nothing more than an *assumption*: inherently, there is nothing wrong with the causal process in Figure 7(b), and we could in fact easily imagine that, because fever is one of the earliest symptoms of pneumonia, this process is actually a better model of the real world than that in Figure 7(a). So why do we choose to eliminate precisely these processes, in order to regain our uniqueness result?

To explain our motivation for this, we need to go back to the analysis of Section 4. There, we considered *timed* vocabularies, in which ground atoms are intended to represent properties at some particular point in time. We then proved that each temporal theory without negation has an execution model that follows its timing, i.e., in which events happen in the right order. As we remarked, such a theory may also have other execution models, in which events happen in the wrong order, but this is not a problem, because all execution models of a positive theory generate the same probability distribution anyway. For theories with negation, however, the situation is more complicated. In that case, we can have three different kinds of execution models: those which follow the timing; those which do not follow the timing, but nevertheless generate the same probability distribution as the ones that do; and those which do not follow the timing and also generate a different probability distribution. The right way to resolve the ambiguity for these theories is obviously to reject this last kind of execution model.

As we will prove in Section 6.3, temporal precedence will do precisely this—at least, if the CP-laws containing negation are not instantaneous. Intuitively, this can be explained as follows. For such a non-instantaneous CP-law, the timings of the atoms in its precondition are strictly earlier than those of its effects. Therefore, in a process which follows this timing, all events which cause one of these atoms must happen before the CP-law itself fires. This is now precisely what temporal precedence assumes. The following example is a variant—or rather, a refinement—of Example 6, which illustrates this.

Example 7
A patient enters the hospital, possibly suffering from pneumonia. At this time, he will be examined by a physician, who will decide to treat the patient with probability 0.95 if he actually has pneumonia. If the patient has pneumonia but the doctor does not treat him, there is a probability of 0.7 that the patient will exhibit a fever by the next morning. We introduce the following propositions:

- *Pneumonia*: "the patient has pneumonia when entering the hospital";
- *Treatment*: "the patient is treated upon admission";
- *Fever*: "the patient has a fever the next morning".

Under this interpretation of our vocabulary, the CP-theory of Example 6 respects the timing and correctly models the example. Clearly, the intended model of the theory is now that of Figure 7(a): in this model, CP-laws fire in the right temporal

order. The process of Figure 7(b), on the other hand, goes against the flow of time ("treatment upon admission" is only caused *after* "fever the next morning"), which should be impossible.

So, as this example illustrates, if our CP-theory respects some intended timing such that the CP-laws containing negation are non-instantaneous, the temporal precedence assumption will resolve the ambiguity in the right way, i.e., by selecting precisely those processes that follow the intended timing. We will now formally define temporal precedence and prove afterwards, in Section 6.3 that this property holds in general.

We start by introducing some mathematical machinery. The basic idea is that a CP-law should only fire after all events that might still affect the truth of its precondition have already happened, i.e., this precondition should not merely be *currently* true, but should in fact already be guaranteed to also remain true in all potential *future* states. This naturally leads to a three-valued logic, where we have truth values $\mathbf{t}$ (guaranteed to remain true), $\mathbf{f}$ (guaranteed to remain false), and $\mathbf{u}$ (still subject to change). Recall that a three-valued interpretation ν is a mapping from the ground atoms of our vocabulary to the set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, which induces for each formula ϕ a truth value ϕ^ν.

Now, if our probabilistic process is in a state s, then the atoms of which we are already sure that they are true are precisely those in $\mathcal{I}(s)$. To figure out which atoms are still unknown, we need to look at which CP-laws might still fire, i.e., at those rules r, for which $body(r)^\nu \neq \mathbf{f}$. Whenever we find such a rule, we know that the atoms in $head(r)$ might all still be caused and, as such, they must be at least unknown. We will now look at a derivation sequence, in which we start by assuming that everything that is currently not $\mathbf{t}$ is $\mathbf{f}$ and then gradually build up the set of unknown atoms by applying this principle.

Definition 10 (*hypothetical derivation sequence*)
A *hypothetical derivation sequence* in a node s is a sequence $(\nu_i)_{0 \leq i \leq n}$ of three-valued interpretations that satisfied the following properties. Initially, ν_0 assigns $\mathbf{f}$ to all atoms not in $\mathcal{I}(s)$. For each $i > 0$, there must be a rule r with $body(r)^{\nu_i} \neq \mathbf{f}$, such that, for all $p \in head_{At}(r)$ with $\nu_i(p) = \mathbf{f}$, it is the case that $\nu_{i+1}(p) = \mathbf{u}$, while for all other atoms p, $\nu_{i+1}(p) = \nu_i(p)$.

Such a sequence is *terminal* if it cannot be extended. A crucial property is now that all such sequences reach the same limit.

Theorem 4
Every terminal hypothetical derivation sequence reaches the same limit, i.e., if $(\nu_i)_{0 \leq i \leq n}$ and $(\nu'_i)_{0 \leq i \leq m}$ are such sequences, then $\nu_n = \nu'_m$.

Proof
Proof of this theorem is given in Section A.1. □

For a state s in a probabilistic process, we will denote this unique limit as ν_s and refer to it as the *potential* in s. Such a ν_s now provides us with an estimate

of which atoms might still be caused, given that we are already in state s. We can now tell whether the part of the process that determines the truth of a formula ϕ has already finished by looking at ν_s; indeed, we can consider this process to be finished iff $\phi^{\nu_s} \neq \mathbf{u}$. We now extend the concept of an execution model to arbitrary CP-theories as follows.

Definition 11 (*execution model—general case*)
Let C be a CP-theory in vocabulary Σ, $\mathcal{T}$ a probabilistic Σ-process, and X an interpretation of the exogenous predicates of C. $\mathcal{T}$ is an *execution model* of C in context X iff

- $\mathcal{T}$ satisfies the conditions of Definition 5 (execution model–positive case);
- For every node s, $body(\mathcal{E}(s))^{\nu_s} \neq \mathbf{u}$, with ν_s the potential in s.

From now on, we will refer to a probabilistic Σ-process that satisfies the original conditions of Definition 5, but not necessarily the additional condition imposed above, as a *weak execution model*.

In the case of Example 6, this definition indeed gives us the result described above, i.e., the process in Figure 7(a) is an execution model of the example, while the one in Figure 7(b) is not. Indeed, if we look at the root $\perp$ of this tree, with $\mathcal{I}(\perp) = \{Pneumonia\}$, we see that we can construct the following terminal hypothetical derivation sequence:

- ν_0 assigns $\mathbf{f}$ to $Treatment$ and $Fever$;
- ν_1 assigns $\mathbf{u}$ to $Treatment$;
- ν_2 assigns $\mathbf{u}$ to $Fever$, because $(\neg Treatment \wedge Pneumonia)^{\nu_1} = \mathbf{u}$;

As such, the only CP-law that can initially fire is the one by which the patient might receive treatment. Afterwards, in every descendant s of $\perp$, $\nu_s(Treatment)$ will be either $\mathbf{t}$ or $\mathbf{f}$. In the branch where it is $\mathbf{f}$, the event by which the patient gets fever because of untreated pneumonia will subsequently happen.

The temporal precedence assumption imposes a constraint on the order in which CP-laws can fire and hence, on the order in which atoms can be caused to become true. In the case of Example 6, this order is fixed and can easily be derived from the syntactical structure of the CP-theory. This is not always the case. As the following example illustrates, the order of events may depend on the context in which they happen.

Example 8
A software system consists of two servers that provide identical services. One server acts as master and the other as slave, and these roles are assigned randomly. Clients can request services. The master makes a selection among these request and the

slave fulfills the requests that are not accepted by the master.

$$(Master(S1) : 0.5) \vee (Slave(S1) : 0.5). \tag{19}$$

$$Master(S2) \leftarrow Slave(S1). \tag{20}$$

$$Slave(S2) \leftarrow Master(S1). \tag{21}$$

$$\forall x \forall s(Accepts(x, s) : 0.6) \leftarrow Application(s) \wedge Master(x). \tag{22}$$

$$\forall x \forall s\ Accepts(x, s) \leftarrow Application(s) \wedge Slave(x) \wedge Master(y) \wedge \neg Accepts(y, s). \tag{23}$$

In all causal processes that satisfy the temporal precedence assumption, the master accepts services before the slave does. However, because who is slave and who is master depends on the result of event (19), this means that we cannot say up-front which of the atoms $Accepts(S1, s)$ and $Accepts(S2, s)$ will be caused first. This shows that the temporal precedence assumption induces a *context dependent stratification* on both events and atoms.

The temporal precedence assumption is correct for many theories—including, as we will prove later, all those temporal theories in which CP-laws containing negation are not instantaneous—but not for all.

Example 9
We consider a variant of the problem of Example 8 in which the slave does not have to wait for the decision of the master, but is allowed to accept any request provided it has not yet been accepted by the master. It is then possible that first the slave and later the master accept the same request, in which case the service is provided two times.

Unlike Example 8, this specification is an *incomplete* description of a probability distribution. Indeed, the probability of a request being handled by the slave now also depends on the probability of the slave reaching a decision before the master does, which is not specified. If we try to model this example with the same CP-theory as Example 8, the temporal precedence assumption would make one particular assumption about the relative speed of the two servers, namely, that the slave is always slower than the master. If we want to model some other distribution, where the slave is sometimes faster than the master, we have to use a different representation style, which allows such information to be incorporated. This will be discussed later in Example 13.

What this discussion illustrates is that, ultimately, it is the responsibility of the user to design his CP-theory in such a way that the intended causal processes satisfy the temporal precedence assumption.

6 Discussion

We now check whether the way in which the previous section has extended the concept of an execution model to cope with negation indeed satisfies the goals that we originally stated.

6.1 The case of positive theories

First of all, we remark that, for positive CP-theories, the new definition (Def. 11) simply coincides with the original one (Def. 5), i.e., for positive theories, there is no difference between execution models and weak execution models. Indeed, because, according to our original definition, it must be the case that $\mathcal{I}(s) \models \mathcal{E}(s)$ for each non-leaf node s, this is an immediate consequence of the following theorem, which follows trivially from the fact that throughout a hypothetical derivation sequence, the truth of an atom p can only increase.

Theorem 5
Let s be a node in a probabilistic Σ-process. For any positive formula ϕ, if $\mathcal{I}(s) \models \phi$, then $\nu_s(\phi) = \mathbf{t}$.

We conclude that, for positive CP-theories, the new definition is simply equivalent to the old one.

6.2 Uniqueness theorem regained

Second, the uniqueness theorem now indeed extends beyond positive theories.

Theorem 6 (*Uniqueness—general case*)
Let C be a CP-theory and X an interpretation of the exogenous predicates of C. If $\mathcal{T}$ and $\mathcal{T}'$ are execution models of C in context X, i.e., $\mathcal{T} \models_X C$ and $\mathcal{T}' \models_X C$, then $\pi_{\mathcal{T}} = \pi_{\mathcal{T}'}$.

Proof
Proof of this theorem is given in Section A.1. □

6.3 Correctness of temporal precedence in temporal CP-theories

In the previous section, we introduced the temporal precedence assumption as a way of solving an ambiguity problem, namely the fact that different weak execution models of a CP-theory with negation might produce different probability distributions. We showed that this assumption was satisfied in the temporal CP-theory of Example 7. We now prove that the temporal precedence assumption is satisfied in a broad class of CP-theories, namely, in all temporal CP-theories in which events containing negation are non-instantaneous. To be more concrete, we will show that if a weak execution model follows the timing of the vocabulary, it also satisfies the temporal precedence assumption.

Our first step is to refine our notion of a theory respecting a timing (Definition 8), to make a distinction between those atoms from some $body_{At}(r)$ that appear only in a *positive* context and those which occur at least once in a *negative* context. The set of all the latter atoms will be denoted as $body^{-}_{At}(r)$, whereas that of all the former ones is $body^{+}_{At}(r)$[4].

[4] Formally, we define, for all sentences ϕ, the sets $At^{+}(\phi)$ and $At^{-}(\phi)$ by simultaneous induction as:

Definition 12 (*strictly respecting a timing*)
A CP-theory C (with negation) *strictly* respects a timing λ if, for all ground atoms h and b:

- If there is CP-law r with $h \in head_{At}(r)$ and $b \in body^+_{At}(r)$, then $\lambda(h) \geq \lambda(b)$;
- If there is CP-law r with $h \in head_{At}(r)$ and $b \in body^-_{At}(r)$, then $\lambda(h) > \lambda(b)$;

Notice that we impose a stronger condition on negative conditions than on positive ones: the times of negative conditions should be *strictly* less than any caused atom. This condition entails that CP-laws with negation are not instantaneous.

Theorem 7
Let C be a CP-theory which strictly respects a timing λ. Every weak execution model of C that follows λ also satisfies temporal precedence and is, therefore, an execution model of C. Moreover, such a process always exists.

Proof
Proof of this theorem is given in Section A.3. □

Intuitively, the theorem states that any causal process of C that is physically possible (i.e., in which no event is caused by conditions that arise only in the future), automatically satisfies temporal precedence. Hence, in the context of CP-theories that strictly respect some intended timing, the temporal precedence assumption applies naturally.

Example 10
We consider a time line divided into a number of different time slots, as illustrated in Figure 8. In the first time slot, a client sends a request to a server. If the server receives a request, then with probability 0.5, he accepts it and sends a reply, all within the same time slot as that in which he received the request. If the client has sent a request and has not received a reply at the end of the time slot, he will repeat his request. A message that is sent has a probability of 0.8 of reaching the recipient in the same time slot as it was sent; with probability 0.1, it reaches the recipient only in the next slot; with the remaining probability of 0.1, it will be lost.

- For $p(\mathbf{t})$ a ground atom, $At^-(p(\mathbf{t})) = \{\}$ and $At^+(p(\mathbf{t})) = \{p(\mathbf{t})\}$;
- For $\phi \circ \psi$, with $\circ$ either $\vee$ or $\wedge$, $At^+(\phi \circ \psi) = At^+(\phi) \cup At^+(\psi)$ and $At^-(\phi \circ \psi) = At^-(\phi) \cup At^-(\psi)$;
- For $\neg\phi$, $At^+(\neg\phi) = At^-(\phi)$ and $At^-(\neg\phi) = At^+(\phi)$;
- For $\Theta x\ \phi$, with Θ either $\forall$ or $\exists$, $At^+(\Theta x\ \phi) = \cup_{t \in H_U(\Sigma)} At^+(\phi[x/t])$ and $At^-(\Theta x\ \phi) = \cup_{t \in H_U(\Sigma)} At^-(\phi[x/t])$, where $H_U(\Sigma)$ is the Herbrand universe.

We can then define $body^-_{At}(r) = At^-(body(r))$ and $body^+_{At}(r) = body_{At}(r) \setminus body^-_{At}(r)$.

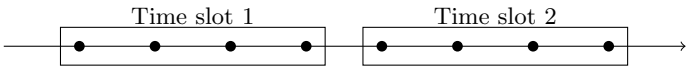


Fig. 8. A division into time slots.

$$(Send(Client, Req, Server, 1) : 0.7). \tag{24}$$

$$\forall t\ (Accept(t) : 0.5) \vee (Reject(t) : 0.5) \leftarrow Recvs(Server, Req, t). \tag{25}$$

$$\forall t\ Send(Server, Answer, Client, t) \leftarrow Accept(t). \tag{26}$$

$$\forall t, s, r, m\ (Recvs(r, m, t) : 0.8) \vee (Recvs(r, m, t+1) : 0.1) \leftarrow Send(s, m, r, t). \tag{27}$$

$$\forall t\ Send(Client, Req, Server, t+1) \leftarrow Send(Client, Req, Server, t) \wedge \neg Recvs(Client, Answer, t). \tag{28}$$

In this CP-theory, (24), (25) and (26) are all instantaneous; the events described by (27) might either take place within one time slot or constitute a propagation to a later time slot, depending on which of the possible effects actually occurs; finally, the events described by (28) all propagate to a later time slot. Because these last events are the only ones in which negation occurs, this theory strictly respects its intended timing and the theorem shows that the semantics gives the intended result.

In summary, a sensible temporal CP-theory should respect its timing. If it strictly respect this timing—that is, the timing is fine-grained enough to make CP-laws with negation non-instantaneous—then all of its weak execution models will automatically satisfy temporal precedence as well. Otherwise, there may be weak execution models that do not satisfy temporal precedence and, hence, will be ruled out as well.

6.4 Validity of a CP-theory

Not all CP-theories have an execution model. Let us illustrate this by the following example.

Example 11
A game is being played between two players, called $White$ and $Black$. If $White$ does not win, this causes $Black$ to win and if $Black$ does not win, this causes $White$ to win.

$$Win(White) \leftarrow \neg Win(Black). \tag{29}$$

$$Win(Black) \leftarrow \neg Win(White). \tag{30}$$

This theory has two weak execution models: one in which (29) fires first and white wins with probability 1, and one in which (30) fires first and black wins with probability 1. However, both of these weak execution models are rejected by the

temporal precedence assumption. Indeed, in each of these weak execution models, it is the case that, for the root $\bot$, $(\neg Win(White))^{\nu_\bot} = \mathbf{u} = (\neg Win(Black))^{\nu_\bot}$, so neither of the two events can happen. So, this is an example of an ambiguity that cannot be resolved by assuming temporal precedence. In order to make a sensible CP-theory out of this example, we would have to add additional information about the probability that one CP-law fires before the other. As will be illustrated in Example 13, such information can be modeled in CP-logic, but requires a different representation style in which temporal arguments are added to the predicates.

Theories which have no execution models are obviously not of interest. This motivates the following definition.

Definition 13 (*valid CP-theories*)
A CP-theory C is *valid* in an interpretation X for its exogenous predicates if it has at least one execution model in context X. If C is valid in all contexts X, we simply say that C is valid.

Clearly, it is only if C is a valid CP-theory that we can associate a probability distribution π_C to it. The theories of Example 6 and Example 8 are valid.

The above discussion raises the question how to recognize whether a theory is valid. We now propose a simple syntactic criterion that guarantees this.

Definition 14 (*stratified CP-theories*)
A CP-theory C is *stratified* if there exists a function λ from the set of its atoms to an interval $[0..n]$ such that C strictly respects λ.

Here, it is possible that the function λ is a timing such as in Section 6.3, but this is not necessary; e.g., it might be the case that λ assigns different natural numbers to atoms that conceptually, in their intended interpretation, are supposed to refer to the same time points. The following corollary of Theorem 7 is of relevance both to temporal and atemporal CP-theories.

Corollary 1
Each stratified theory C has an execution model.

We remark that, in particular, all positive theories are stratified, because, for such a theory, we can simply assign 0 to all ground atoms. An example of a stratified theory containing negation is given in Example 10. The theory of Example 8 is not stratified because the atoms $Accepts(S1, x)$ and $Accepts(S2, x)$ cannot be ordered in time, since the times at which they are made true depends on who is the master. This is an example of a valid but unstratified CP-theory. We therefore conclude that the existence of a stratification is a sufficient condition for the existence of an execution model—and hence of the theory actually defining a probability distribution—but not a necessary one.

6.5 The representation of time in CP-logic

In the preceding sections, we have encountered two quite different styles of knowledge representation: temporal theories explicitly include time, while atemporal theories make abstraction of it.

There may be several reasons for making time explicit. One obvious reason is if we are actually interested in the intermediate states of the process. Other reasons might be that the causal processes in a domain are simply too complex to model without explicit time. Below, we illustrate two such cases.

In CP-logic, each atom starts out as false and might become true during the process; moreover, if at some point an atom becomes true, it will remain true. In applications where the obvious relevant properties of the domain of interest do not behave like this, we cannot simply represent them by atoms in our CP-theory. As already mentioned in Section 4, this problem can typically be solved by explicitly including time in the representation. The following example illustrates how this methodology can be used to handle domains in which there are causes for both a property and its negation.

Example 12
Consider the following variant of Example 2, in which a doctor can now administer a medicine to suppress chest pain with probability 0.9.

$$Pneumonia(1). \tag{31}$$

$$\forall d\ (Pneumonia(d+1) : 0.8) \leftarrow Pneumonia(d). \tag{32}$$

$$\forall d\ (Chestpain(d) : 0.6) \leftarrow Pneumonia(d) \wedge \neg Suppressed(d). \tag{33}$$

$$\forall d\ Medicine(d) \leftarrow Chestpain(d). \tag{34}$$

$$\forall d\ (Suppressed(d+1) : 0.9) \leftarrow Medicine(d). \tag{35}$$

In this representation, the use of negation allows the predicate *Suppressed* to act as a cause for not having chestpain.

We now discuss another type of application that requires time to be made explicit. As mentioned before, temporal precedence might give unintended results for theories which are not temporal or whose granularity of time is such that negation occurs in instantaneous events. In such cases, the obvious solution is to make time explicit and ensure it is fine-grained enough to make all events with negation non-instantaneous[5]. To illustrate, we consider the following refinement of Example 9.

Example 13
We again consider the setting of Example 9, where the slave does not necessarily wait for the decision of the master, before deciding whether to accept the request himself. This might be the case, for instance, in a system where the two servers have not been properly synchronized. As we explained, for such a model to be a complete

[5] For most real world events, there exists, at least in principle, some time scale that would make them non-instantaneous. For instance, even an event such as the temperature of a gas increasing when the space in which it is contained decreases, only manifests itself after the molecules of the gas have travelled a certain microscopic distance, which does take a—small, but in principle non-zero—amount of time. Examples of truly instantaneous events can be found in quantum mechanics (if the state of one object collapses, this instantaneously causes the collapse of the state of each entangled object) and abstract properties defined by social convention (e.g., signing a purchase deed instantaneously makes one the owner of a house).

description of a probability distribution, we then also need to include information about the probability that the slave decides before the master. We will assume that, at each time point where the master has not decided yet, there is a probability of 0.2 that he will decide; for the slave, we assume that this probability is 0.8.

$$(Master(S1) : 0.5) \vee (Slave(S1) : 0.5) \leftarrow .$$

$$Master(S2) \leftarrow Slave(S1).$$

$$Slave(S2) \leftarrow Master(S1).$$

$$\forall x \forall s \forall t(Decides(x, s, t) : 0.2) \leftarrow \begin{array}{l} Master(x) \ \wedge Application(s) \wedge \\ \neg \exists t' \ (t' < t \wedge Decides(x, s, t')). \end{array}$$

$$\forall x \forall s \forall t(Accepts(x, s, t) : 0.6) \leftarrow Master(x) \wedge Decides(x, s, t).$$

$$\forall x \forall s \forall t(Decides(x, s, t) : 0.8) \leftarrow \begin{array}{l} Slave(x) \wedge Application(s) \wedge \\ \neg \exists t' \ (t' < t \wedge Decides(x, s, t')). \end{array}$$

$$\forall x \forall s \forall t \ Accepts(x, s, t) \leftarrow \begin{array}{l} Slave(x) \wedge Decides(x, s, t) \wedge \\ \neg \exists y \exists t'(Master(y) \wedge \ t' < t \wedge Accepts(y, s, t')). \end{array}$$

In this CP-theory, we have introduced the predicate $Decides(x, s, t)$ as a *reification* of the events by which the servers reach their decision (i.e., the events that were described by (22) and (23) in our original theory from Example 8). The meaning of this predicate is that server x makes his decision on application s at time t. The above CP-theory models the situation of an eager slave that decides on applications much faster than the master, which causes many services to be provided twice.

7 The relation to Bayesian networks

In this section, we investigate the relation between CP-logic and Bayesian networks. Before we begin, let us briefly recall the definition of a Bayesian network. Such a network consists of a directed acyclic graph and a number of probability tables. Every node n in the graph represents a random variable, which has some domain $dom(n)$ of possible values. A network B defines a unique probability distribution π_B over the set of all possible assignments $n_1 = v_1, \ldots, n_m = v_m$ of values to all of these random variables, with all $v_i \in dom(n_i)$. First, this π_B must obey a probabilistic independence assumption expressed by the graph, namely, that every node n is probabilistically independent of all of its non-descendants, given its parents. This allows the probability $\pi_B(n_1 = v_1, \ldots, n_m = v_m)$ of such an assignment of values to all random variables to be rewritten as a product of conditional probabilities $\prod_i \pi_B(n_i = v_i \mid pa(n_i) = \mathbf{v})$, where each $pa(n_i)$ is the tuple of all parents of n_i in the graph. The probability tables associated to the network now specify precisely all of these conditional probabilities $\pi_B(n_i = v_i \mid pa(n_i) = \mathbf{v})$. The second condition imposed on π_B is then simply that all of these conditional probabilities must match the corresponding entries in these tables. It can be shown that this indeed suffices to uniquely characterize a single distribution.

Most commonly, Bayesian networks are constructed without any explicit references to time, since this tends to produce the simplest models. However, in some

cases such a representation does not suffice; then, one typically uses a so-called *dynamic Bayesian network* (Ghahramani 1998) which makes time explicit in much the same way as can be done in CP-logic.

Like CP-logic, Bayesian networks are a formal language that can be used to represent causal relations in a domain. This is done by chosing as the parents of a node x all nodes y for which it is the case that the value of y has a direct effect on the value of x. The values in the conditional probability table for x then quantifiy the joint effect that all of these parents together have on x. Bayesian networks constructed in this way are usually called *causal* networks, to dinstinguish them from "non-causal" networks which do not necessarily follow the direction of causal relations. Causal Bayesian network are more informative than non-causal ones: not only do they define a probability distribution, but they also specify what will happen when external action intervenes with the normal operation of the causal mechanisms it describes (Pearl 2000).

In this section, we will compare causal Bayesian networks to CP-logic. We first show that, because Bayesian networks can easily be unfolded into probability trees (Shafer 1996), they can be mapped to CP-logic in a straightforward way. We then discuss how CP-logic differs from Bayesian networks. There are essentially three main differences. Our representation is more fine-grained and modular in the sense that a single probabilistic causal law can express the effect that *some* of the "parents" of an atom have on it, regardless of the effect of others. It is also more qualitative, since we can use first-order formulas to specify in which circumstances the "parents" will have a certain effect on the child, while Bayesian networks encode such information in probability tables. Finally, it is also more general, in the sense that it can directly represent cyclic causal relations, which a Bayesian network cannot. We remark that these comparisons consider only the "vanilla" way of writing down Bayesian networks, i.e., as a drawing of a directed acyclic graph accompanied by tables of numbers. A large number of alternative notations exist in the literature, e.g., (Comley and Dowe 2003). These provide more elegant ways of handling all but one of the "shortcomings" of Bayesian networks that we will mention—the exception being, to the best of our knowledge, their inability to directly represent cyclic causal relations.

After this discussion of representation issues, Section 7.5 will discuss interventions in causal Bayesian networks and show that the semantics of CP-logic induces a natural counterpart to this notion.

7.1 Bayesian networks in CP-logic

As also mentioned in Shafer's book, a Bayesian network can be seen as a description of a class of probability trees. We first make this more precise. To make it easier to compare to CP-logic later on, we will start by introducing a logical vocabulary for describing a Bayesian network.

Definition 15
Let B be a Bayesian network. The vocabulary Σ_B consists of a predicate symbol P_n for each node n of B and a constant C_v for each value v in the domain of n.

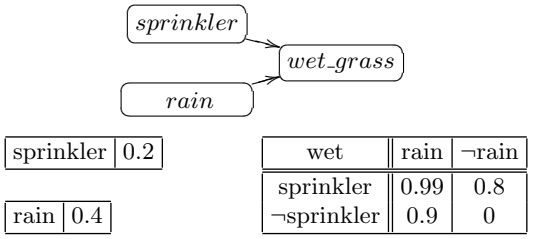


Fig. 9. Bayesian network for the sprinkler example.

Fig. 10. Process corresponding to the sprinkler Bayesian network.

Now, we want to relate a Bayesian network B to a class of Σ_B-processes. Intuitively, we are interested in those processes, where the flow of events follows the structure of the graph and every event propagates the values of the parents of a node to this node itself. We illustrate this by the following famous example.

Example 14 (*Sprinkler*)
The grass can be wet because it has rained or because the sprinkler was on. The probability of the sprinkler causing the grass to be wet is 0.8; the probability of rain causing the grass to be wet is 0.9; and the probability of the grass being wet if both the sprinkler is on and it is raining is 0.99. The *a priori* probability of rain is 0.4 and that of the sprinkler having been on is 0.2.

The Bayesian network formalization of this example can be seen in Figure 9. Figure 10 shows a process that corresponds to this network. Here, we have exploited the fact that all random variables of the Bayesian network are boolean, by representing every random variable by a single atom, i.e., writing for instance Wet and $\neg Wet$ instead of $Wet(True)$ and $Wet(False)$. Formally, we define the following class of processes for a Bayesian network.

Definition 16
Let B be a Bayesian network. A *B-process* is a probabilistic Σ_B-process $\mathcal{T}$ for which there exists a mapping $\mathcal{N}$ from nodes of $\mathcal{T}$ to nodes of B, such that the following conditions are satisfied. For every branch of $\mathcal{T}$, $\mathcal{N}$ is a one-to-one mapping between the nodes on this branch and the nodes of B, which is order preserving, in the sense

that, for all s, s' on this branch, if $\mathcal{N}(s)$ is an ancestor of $\mathcal{N}(s')$ in B, then s must be an ancestor of s' in $\mathcal{T}$. If $\mathcal{N}(s)$ is a node n with domain $\{v_1, \ldots, v_k\}$ and parents $p_1 \ldots, p_m$ in B, then the children of s in $\mathcal{T}$ are nodes $s_1, \ldots, s_k$, for which:

- $\mathcal{I}(s_i) = \mathcal{I}(s) \cup \{P_n(C_{v_i})\}$;
- The edge from s to s_i is labeled with the entry in the table for n, that gives the conditional probability of $n = v_i$ given $p_1 = w_1, \ldots, p_m = w_m$, where each w_i is the unique value from the domain of p_i for which $P_{p_i}(C_{w_i}) \in \mathcal{I}(s)$.

It should be clear that every leaf s of such a B-process $\mathcal{T}$ describes an assignment of values to all nodes of B, i.e., every node n is assigned the unique value v for which $P_n(c_v) \in \mathcal{I}(s)$. Moreover, the probability $\mathcal{P}(s)$ of such a leaf is precisely the product of all the appropriate entries in the various conditional probability distributions. Therefore, the distribution $\pi_{\mathcal{T}}$ coincides with the distribution defined by the network B.

We now construct a CP-theory CP_B, such that the execution models of CP_B will be precisely all B-processes. We first illustrate this process by showing how the Bayesian network in Figure 9 can be transformed into a CP-theory.

Example 14 (*Sprinkler—cont'd*)
We can derive the following CP-theory from the Bayesian network in Figure 9.

$$\begin{aligned}
(Wet : 0.99) &\leftarrow \quad Sprinkler \wedge \quad Rain \\
(Wet : 0.8) &\leftarrow \quad Sprinkler \wedge \neg Rain. \\
(Wet : 0.9) &\leftarrow \neg Sprinkler \wedge \quad Rain. \\
(Wet : 0.0) &\leftarrow \neg Sprinkler \wedge \neg Rain. \\
(Sprinkler : 0.2). & \\
(Rain : 0.4). &
\end{aligned}$$

Again, this example exploits the fact that the random variables are all boolean, by using the more readable representation of Wet and $\neg Wet$ instead of $Wet(True)$ and $Wet(False)$. It should be obvious that the process in Figure 10 is an execution model of this theory and, therefore, that this theory defines the same probability distribution as the Bayesian network.

It is now easy to see that the encoding used in the above example generalizes. Concretely, for every node n with parents $p_1, \ldots, p_m$ and domain $\{v_1, \ldots, v_k\}$, we should construct the set of all rules of the form:

$$(P_n(C_{v_1}) : \alpha_1) \vee \cdots \vee (P_n(C_{v_k}) : \alpha_k) \leftarrow P_{p_1}(C_{w_1}) \wedge \cdots P_{p_m}(C_{w_m}),$$

where each w_i belongs to the domain of p_i and each α_j is the entry for $n = v_j$, given $p_1 = w_1, \ldots, p_m = w_m$ in the CPT for n. Let us denote the CP-theory thus constructed by CP_B. The following result is then obvious.

Theorem 8

Let B be a Bayesian network. Every B-process $\mathcal{T}$ is an execution model of the CP-theory CP_B, i.e., $\mathcal{T} \models CP_B$. Therefore, the semantics of B coincides with the distribution π_C.

This result shows that CP-logic offers a straightforward way of representing Bayesian networks. We now discuss three ways in which it offers more expressivity.

7.2 Multiple causes for the same effect

In a process corresponding to a Bayesian network, the value of each random variable is determined by a single event. CP-logic, on the other hand, allows multiple events to affect the same property. This leads to better representations for effects that have a number of independent causes. Let us illustrate this by the following example.

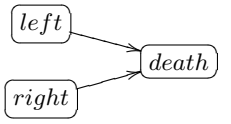
Example 15
We consider a game of Russian roulette that is being played with two guns, one in the player's left hand and one in his right, each of which has a bullet in one of its six chambers.

$$(Death : 1/6) \leftarrow Pull\_trigger(Left\_gun).$$
$$(Death : 1/6) \leftarrow Pull\_trigger(Right\_gun).$$

In this example, there are two "causal mechanisms" that might lead to $Death$: one is the fact that pulling the trigger of the left gun might cause a bullet to hit the person's left temple, and the other is the fact that pulling the trigger of the right gun might cause a bullet to hit the person's right temple. They are *independent* in the following sense: once we know how many and which of these mechanisms are actually activated (i.e., which of the two triggers are pulled), then observing whether one of these possible causes actually results in the effect (i.e., whether one of the bullets is actually fired and kills the persons) provides no information about whether one of the other causes will cause the effect (i.e., whether one of the other bullets is also fired). Mathematically, this is of course saying nothing more than the probability of the effect occurring should be equal to the result of applying a *noisy-or*[6] to the multiset of the probabilities with which each of the causal mechanisms that are actually activated causes the effect, i.e., if both guns are fired, the probability of death should be $1 - (1 - 1/6)^2$. This independence is precisely the condition that is required in order to be able to represent each of these two causal mechanism by a separate CP-law, as in the above example. To succinctly describe this situation, we say that $Pull\_trigger(Left\_gun)$ and $Pull\_trigger(Right\_gun)$ are *independent causes* for $Death$.

For instance, in Example 14, $Rain$ and $Sprinkler$ were not independent causes for Wet, since the probability of Wet given both $Rain$ and $Sprinkler$ is 0.99, which is not equal to $1 - (1 - 0.8)(1 - 0.9) = 0.98$. In this case, the causal mechanisms

[6] The *noisy-or* maps a multiset of probabilities α_i to $1 - \prod_i (1 - \alpha_i)$.



death	left	¬left
right	11/36	1/6
¬right	1/6	0

Fig. 11. A Bayesian network for Example 15 .

by which *Rain* and *Sprinkler* cause Wet therefore appear to reenforce each other: it might be that a light drizzle only causes the grass to get slightly moist, and that sometimes the pressure on the water main is so low that the sprinkler by itself cannot get the grass really wet, but that a light drizzle *and* a lightly spraying sprinkler *together* would be enough to cause Wet, even though neither of them separately would do the trick. Because *Sprinkler* and *Rain* are not independent causes in this example, we cannot use a representation of the form:

$$(Wet : \alpha) \leftarrow Sprinkler.$$
$$(Wet : \beta) \leftarrow Rain.$$

and instead have no choice but to use the representation shown on page 32.

Figure 11 shows a Bayesian network for the Russian roulette example. The most obvious difference between this representation and ours concerns the independence between the two different causes for death. In the CP-theory, this independence is expressed by the *structure* of the theory, whereas in the Bayesian network, it is a *numerical* property of the probabilities in the conditional probability table for *Death*. Because of this, the CP-theory is more elaboration tolerant, since adding or removing an additional cause for *Death* simply corresponds to adding or removing a single CP-law. Moreover, its representation is also more compact, requiring, in general, only n probabilities for n independent causes, instead of the 2^n entries that are needed in a Bayesian network table. Of course, these entries are nothing more than the result of applying a *noisy-or*[7] to the multiset of the probabilities with which each of the causes that are present actually causes the effect.

In graphical modeling, it is common to consider variants of Bayesian networks, that use more sophisticated representations of the required conditional probability distributions than a simple table. Including the *noisy-or* as a structural element in such a representation achieves the same effect as CP-logic when it comes to representing independent causes.

[7] The *noisy-or* maps a multiset of probabilities α_i to $1 - \prod_i (1 - \alpha_i)$.

7.3 First-order logic representation of causes

In CP-logic, the cause of an event can be represented in a qualitative way, by means of a first-order formula. Bayesian networks, on the other hand, encode such information in the probability tables.

Example 16
In the so-called *Wumpus world*, an agent moves through a grid, which contains, among other things, a number of bottomless pits. One aspect of this world is that if a position x is next to such a pit, then with a certain probability α, a breeze can be felt there (often, α is simply taken to be 1). In CP-logic, we could write the following CP-law:

$$\forall x\ (Breeze(x) : \alpha) \leftarrow \exists y\ NextTo(x, y) \wedge Pit(y).$$

For a grid in which square A is surrounded by squares B, C, D and E, a Bayesian network could represent the effect of $Pit(B), Pit(C), Pit(D), Pit(E)$ on $Breeze(A)$ by the following table:

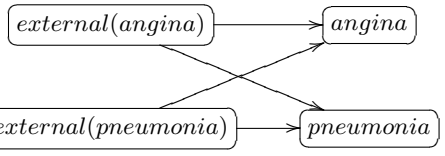
				$Breeze(A)$
$Pit(B)$	$Pit(C)$	$Pit(D)$	$Pit(E)$	α
$Pit(B)$	$Pit(C)$	$Pit(D)$	$\neg Pit(E)$	α
		$\vdots$		$\vdots$
$\neg Pit(B)$	$\neg Pit(C)$	$\neg Pit(D)$	$Pit(E)$	α
$\neg Pit(B)$	$\neg Pit(C)$	$\neg Pit(D)$	$\neg Pit(E)$	0

In this example, CP-logic offers a representation which is considerably more concise than that of the Bayesian network. This manifests itself in two ways: first, our first-order representation succeeds in defining the probability of $Breeze(x)$ for *all* squares x simultaneously by a single CP-law, while each square would need its own (identical) probability table in the Bayesian network; second, it can also summarize the 2^4 entries that make up the probability table for each $Breeze(x)$ by the single first-order precondition of this CP-law. Again, these shortcomings have already been recognized by the Bayesian network community, leading to, for instance, the use of decision trees to represent probability tables (Comley and Dowe 2003), various forms of *parameter tying* and first-order versions of Bayesian networks such as Bayesian Logic Programs (Kersting and De Raedt 2000) (see also Section 9.4.3).

We remark that this feature of CP-logic cannot really be seen separately from that discussed in the previous section: it is precisly because we split up the effect that "parents" have on their "child" into a number of independent causal laws, that we get more opportunity to exploit the expressivity of our first-order representation.

7.4 Cyclic causal relations

In real life, probabilistic processes may consist of events that might propagate values in opposite directions. We already saw this in Example 2, where angina could cause pneumonia, but, vice versa, pneumonia could also cause angina. In CP-logic, such



angina	e(p)	¬e(p)
e(a)	1	1
¬e(a)	0.2	0

pneumonia	e(p)	¬e(p)
e(a)	1	0.3
¬e(a)	1	0

Fig. 12. Bayesian network for the *angina-pneumonia* causal loop.

causal loops do not require any special treatment. For instance, the loop formed by the two CP-laws

$$(Angina : 0.2) \leftarrow Pneumonia.$$
$$(Pneumonia : 0.3) \leftarrow Angina.$$

correctly behaves as follows:

- If the patient has neither angina nor pneumonia by an external cause ('external' here does not mean exogenous, but simply that this cause is not part of the causal loop), then he will have neither;
- If the patient has angina by an external cause, then with probability 0.3 he will also have pneumonia;
- If the patient has pneumonia by an external cause, then with probability 0.2 he will also have angina;
- If the patient has both pneumonia and angina by an external cause, then he will obviously have both.

In order to get the same behaviour in a Bayesian network, this would have to be explicitly encoded. For instance, one could introduce new, artificial random variables *external*(*angina*) and *external*(*pneumonia*) to represent the possibility that *angina* and *pneumonia* result from an external cause and construct the Bayesian network that is shown in Figure 12. In general, to encode a causal loop formed by n properties, one would introduce n additional nodes, i.e., all of the n original properties would have the same n artificial nodes as parents.

7.5 Inverventions in CP-logic

Pearl's work investigates the behaviour of causal models in the presence of *interventions*, i.e., outside manipulations that preempt the normal behaviour of the system. His key observation here is that causal relations are robust, in the sense that, even when *some* causal relations are intervened with, the *other* causal relations will still hold as before. Formally, an intervention for Pearl is something of the form $do(X = x)$ that sets the value of a random variable X to x. In doing this, all edges leading into X are removed, because even though they represent the causal

mechanism that would *normally* determine the value of X, they become irrelevant when the value of X is determined by an external intervention instead.

It is also possible to consider such interventions in the context of CP-logic. Our representation of a causal system is a modular one, in which the atomic unit is a single CP-law. Because of this, our language comes with a specific kind of interventions "built-in": if we want to know what the result is of intervening with a single causal law r, we can simply consider the theory from which this one law is removed (and possibly replaced by some other law). So, to judge the effect of doing an intervention that prevents r, we simply have to look at $\pi_{C\setminus\{r\}}$ instead of π_C, in (roughly) the same way that Pearl would look at something of the form $P(\cdot \mid do(X = x))$ instead of $P(\cdot)$. The opposite is of course also possible: if we want to know the effect of doing an intervention that instead establishes an additional causal law r, we could look at $\pi_{C\cup\{r\}}$.

To illustrate, let us consider another medical example. A tumor in a patient's kidney might cause kidney failure, which might cause the death of the patient; however, to make matters even worse, the tumor can also metastasize to the brain, which might also, independently, kill the patient. We can represent this as:

$$\begin{aligned}(KidneyFailure : 0.1) &\leftarrow KidneyTumor.\\ (BrainTumor : 0.1) &\leftarrow KidneyTumor.\\ (Death : 0.5) &\leftarrow BrainTumor.\\ (Death : 0.9) &\leftarrow KidneyFailure.\end{aligned}$$

Now, let us suppose that we want to know what the effect will be of putting the patient on a dialysis machine, which allows him to survive kidney failure. To anwer this question, we simply remove the last of these causal laws (since the dialysis is precisely meant to prevent this particular causality from taking effect) and look at the semantics of the resulting theory. If, say, the dialysis also carries some small risk, we can also add new causal laws such as:

$$(Death : 0.01) \leftarrow Dialysis.$$

In this way, the semantics of CP-logic already carries within it a notion of intervention, which is also slightly more finegrained than that of Pearl, since we can consider interventions that prohibit a single causal law (as in the above example), whereas Pearl only considers interventions that prohibit *all* causal laws that affect the value of a certain random variables. In the case of the above example, therefore, we would have to either intervene to prevent *all* possible causes for death, including the brain tumor, or none at all. Admittedly, it is of course easy enough to solve this problem by introducing some intermediate variable, say "high levels of toxins in the blood", between kidney failure and death.

Among other things, Pearl uses interventions to make sense of the statistical phenomenon known as *Simpson's paradox*. Because this is somewhat of a benchmark for causal formalisms, we will briefly discuss how CP-logic can deal with it.

Simpson's paradox refers to the phenomenon that a population can sometimes be partitioned in such a way that a certain outcome has a low probability in each of the partitioning sets, yet has a high probability in the population over all (or vice versa). For instance, a certain drug might be harmful to both men and women, but appear beneficial for persons of unknown sex. More precisely, taking the drug and recovering might be positively correlated in the population at large, but become negatively correlated when conditioning on sex. This can happen, for instance, if men are both more likely to take the drug *and* to spontaneously recover from the disease. (Because then observing that a patient takes the drug increases the probability that he is male, which in turn increases the probability that he will recover on his own, thus erroneously suggesting that the drug had some benefical effects on this recovery.)

The crux of Simpson's paradox is that the same probability distribution can be generated by different sets of causal laws, and that in order to figure out whether, e.g., some drug has a positive effect on a patient's condition, it are really these causal laws that matter. Pearl's book shows that the paradox can therefore be resolved by considering the causal models behind the probability distribution. In CP-logic, we can do the same. Let us illustrate this by a famous real-world example: it was found that women had a significantly lower acceptance rate than men for the graduate school of the University of California at Berkeley, which led to a discrimination law-suit against the university. However, it turned out that none of the individual departments of the university had a lower acceptance rate for women than for men; instead, it was simply the case that women were significantly more likely to apply to departments with a low acceptance rate.

A highly simplified model of the real situation might therefore have looked something like this:

$$\forall x\ (Apply(x, Engineering) : 0.7) \vee (Apply(x, Literature) : 0.3) \leftarrow Man(x).$$
$$\forall x\ (Apply(x, Engineering) : 0.2) \vee (Apply(x, Literature) : 0.8) \leftarrow Woman(x).$$
$$\forall x\ (Accepted(x) : 0.6) \leftarrow Apply(x, Engineering).$$
$$\forall x\ (Accepted(x) : 0.3) \leftarrow Apply(x, Literature).$$

Here, there clearly is no gender discrimination: the gender of the applicant plays no role in the CP-laws that describe how the university decides whether to accept an application. The reason for the law suit is that, if we only look at the acceptance rates of men and women for the university as a whole, we cannot distinguish this CP-theory from, e.g., the following one:

$$\forall x\ (Apply(x, Engineering) : 0.6) \vee (Apply(x, Literature) : 0.5) \leftarrow Man(x).$$
$$\forall x\ (Apply(x, Engineering) : 0.4) \vee (Apply(x, Literature) : 0.6) \leftarrow Woman(x).$$
$$\forall x\ (Accepted(x) : 0.3) \leftarrow Apply(x, Engineering) \wedge Woman(x).$$
$$\forall x\ (Accepted(x) : 0.6) \leftarrow Apply(x, Engineering) \wedge Man(x).$$
$$\forall x\ (Accepted(x) : 0.4) \leftarrow Apply(x, Literature) \wedge Woman(x).$$
$$\forall x\ (Accepted(x) : 0.5) \leftarrow Apply(x, Literature) \wedge Man(x).$$

Indeed, both theories yield an acceptance rate of 0.36 for women and an acceptance rate of 0.51 for men. In the law suit, the acceptance rates of individual departments were used to argue that the first model was, in fact, the correct one. Purely theoretically, another option could have been to conduct a randomized experiment to eliminate selection bias: instead of allowing students to choose their department, we would assign it at random. In CP-logic terms, this corresponds to the intervention of removing the first two CP-laws from both theories and replacing them by:

$$\forall x\ (Apply(x, Engineering) : 0.5) \vee (Apply(x, Literature) : 0.5).$$

If we were to perform this intervention, the first theory would predict a new acceptance rate of 0.45 for men and women alike, whereas the second would predict an acceptance rate of 0.35 for women and 0.55 for men. So, we see here that the kind of interventions induced by the semantics of CP-logic is able to explain Simpson's paradox, and does so in essentially the same way that Pearl does.

8 CP-logic and logic programs

There is an obvious similarity between the syntax of CP-logic and that of logic programs. Moreover, the constructive processes that we used to define the semantics of a CP-theory are also similar to the kind of fixpoint constructions used to define certain semantics for logic programs. In this section, we will investigate these similarities. To be more concrete, we will first define a straightforward probabilistic extension of logic programs, called *Logic Programs with Annotated Disjunctions*, and then prove that this is essentially equivalent to CP-logic.

The connection between causal reasoning and logic programming has long been implicitly present; we can refer in this respect to, for instance, formalizations of situation calculus in logic programming (Pinto and Reiter 1993; Van Belleghem et al. 1997). Here, we now make this relation explicit, by showing that the language of CP-logic, that we have constructed directly from causal principles, corresponds to existing logic programming concepts. In this respect, our work is similar to that of (McCain and Turner 1996), who defined the language of causal theories, which was then shown to be closely related to logic programming. However, as we will discuss later, McCain and Turner formalise somewhat different causal intuitions, which leads to a correspondence to a different logic programming semantics. Our

results from this section will help to clarify the position of CP-logic among related work in the area of probabilistic logic programming, such as Poole's Independent Choice Logic (Poole 1997). Moreover, they provide additional insight into the role that causality plays in such probabilistic logic programming languages, as well as in normal and disjunctive logic programs.

8.1 Logic Programs with Annotated Disjunctions

In this section, we define the language of *Logic Programs with Annotated Disjunctions*, or *LPADs* for short. This is a probabilistic extension of logic programming, which is based on disjunctive logic programs. This is a natural choice, because disjunctions themselves—and therefore also disjunctive logic programs—already represent a kind of uncertainty. Indeed, to give just one example, we could use these to model indeterminate effects of actions. Consider, for instance, the following disjunctive rule:

$$Heads \vee Tails \leftarrow Toss.$$

This offers a quite intuitive representation of the fact that tossing a coin will result in either heads or tails. Of course, this is not all we know. Indeed, a coin also has equal probability of landing on heads or tails. The idea behind LPADs is now simply to express this by annotating each of the disjuncts in the head with a probability, i.e., we write:

$$(Heads : 0.5) \vee (Tail : 0.5) \leftarrow Toss.$$

Formally, an LPAD is a set of rules:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \phi, \tag{36}$$

where the h_i are ground atoms and ϕ is a sentence. As such, LPADs are syntactically identical to CP-logic. However, we will define their semantics quite differently. For instance, the above example will express that precisely one of the following logic programming rules holds: either $Heads \leftarrow Toss$ holds, i.e., if the coin is tossed this will yield heads, or the rule $Tails \leftarrow Toss$ holds, i.e., tossing the coin gives tails. Each of these two rules has a probability of 0.5 of being the actual instantiation of the disjunctive rule.

More generally, every rule of form (36) represents a probability distribution over the following set of logic programming rules:

$$\{(h_i \leftarrow \phi) \mid 1 \leq i \leq n\}.$$

From these distributions, a probability distribution over logic programs is then derived. To formally define this distribution, we introduce the following concept of a *selection*. We use the notation $head^*(r)$ to denote the set of pairs $head(r) \cup \{(\emptyset, 1 - \sum_{(h:\alpha)\in head(r)} \alpha)\}$, where $\emptyset$ represents the possibility that none of the h_i's are caused by the rule r.

Definition 17 (*C-selection*)
Let C be an LPAD. A *C-selection* is a function σ from C to $\bigcup_{r\in C} head^*(r)$, such

that for all $r \in C$, $\sigma(r) \in head^*(r)$. By $\sigma^h(r)$ and $\sigma^\alpha(r)$ we denote, respectively, the first and second element of the pair $\sigma(r)$. The set of all C-selections is denoted as $\mathcal{S}_C$.

The probability $P(\sigma)$ of a selection σ is now defined as $\prod_{r \in C} \sigma^\alpha(r)$. For a set $S \subseteq \mathcal{S}_C$ of selections, we define the probability $P(S)$ as $\sum_{\sigma \in S} P(\sigma)$. By C^σ we denote the logic program that consists of all rules $\sigma^h(r) \leftarrow body(r)$ for which $r \in C$ and $\sigma^h(r) \neq \emptyset$. Such a C^σ is called an *instance* of C. We will interpret these instances by the well-founded model semantics. Recall that, in general, the well-founded model of a program P, $wfm(P)$, is a pair (I, J) of interpretations, where I contains all atoms that are certainly true and J contains all atoms that might possibly be true. If $I = J$, then the well-founded model is called exact. Intuitively, if $wfm(P)$ is exact, then the truth of all atoms can be decided, i.e., everything that is not false can be derived. In the semantics of LPADs, we want to ensure that all uncertainty is expressed by means of the annotated disjunctions. In other words, given a specific selection, there should no longer be any uncertainty. We therefore impose the following criterion.

Definition 18 (*soundness*)
An LPAD C is *sound* iff all instances of C have an exact well-founded model.

For such LPADs, the following semantics can now be defined.

Definition 19 (*instance based semantics* μ_C)
Let C be a sound LPAD. For an interpretation I, we denote by $W(I)$ the set of all C-selections σ for which $wfm(C^\sigma) = (I, I)$. The *instance based semantics* μ_C of C is the probability distribution on interpretations, that assigns to each I the probability $P(W(I))$ of this set of selections $W(I)$.

It is straightforward to extend this definition to allow for exogenous predicates as well. Indeed, in Section 2.2, we have already seen how to define the well-founded semantics for rule sets with open predicates, and this is basically all that is needed. Concretely, given an interpretation X for a set of exogenous predicates, we can define the instance based semantics μ_C^X given X as the distribution that assigns, to each interpretation I of the endogenous predicates, the probability of the set of all selections σ for which (I, I) is the well-founded models of C^σ given X. Of course, this semantics is only defined for LPADs that are sound in X, meaning that the well-founded model of each C^σ given X is two-valued.

8.2 Equivalence to CP-logic

Every CP-theory is syntactically also an LPAD and vice versa. The key result of this section is that the instance based semantics μ_C for LPADs coincides with the CP-logic semantics π_C defined in Sections 3 and 5.

Theorem 9
Let C be a CP-theory that is valid in X. Then C is also an LPAD that is sound in X and, moreover, $\mu_C^X = \pi_C^X$.

Proof
Proof of this theorem is given in Section A.2. □

We remark that it is not the case that every sound LPAD is also a valid CP-theory. In other words, there are some sound LPADs that cannot be seen as a sensible description of a set of probabilistic causal laws.

Example 17
It is easy to see that the following CP-theory has no execution models.

$$(P : 0.5) \vee (Q : 0.5) \leftarrow R.$$
$$R \leftarrow \neg P.$$
$$R \leftarrow \neg Q.$$

However, each of its instances has an exact well-founded model: for $\{P \leftarrow R; R \leftarrow \neg P; R \leftarrow \neg Q\}$ this is $\{R, P\}$ and for $\{Q \leftarrow R; R \leftarrow \neg P; R \leftarrow \neg Q\}$ this is $\{R, Q\}$. Clearly, this CP-theory does not have execution models that satisfy the temporal precedence assumption.

8.3 Discussion

The results of this section relate CP-logic to LPADs and, more generally speaking, to the area of logic programming and its probabilistic extensions. As such, these results help to position CP-logic among related work, such as Poole's Independent Choice Logic and McCain and Turner's causal theories, which we will discuss in Section 9.2. Moreover, they also provide a valuable piece of knowledge representation methodology for these languages, by clarifying how they can represent causal information. To illustrate, we now discuss the relevance of our theorem for some logic programming variants.

Disjunctive logic programs. In probabilistic modeling, it is often useful to consider the qualitative structure of a theory separately from its probabilistic parameters. Indeed, for instance, in machine learning, the problems of structure learning and parameter learning are two very different tasks. If we consider only the structure of a CP-theory, then, syntactically speaking, we end up with a *disjunctive logic program*, i.e., a set of rules:

$$h_1 \vee \cdots \vee h_n \leftarrow \phi. \tag{37}$$

We can also single out the qualitative information contained in the semantics π_C of such a CP-theory. Indeed, as we have already seen, like any probability distribution over interpretations, π_C induces a possible world semantics, consisting of those interpretations I for which $\pi_C(I) > 0$. Thus we can define:

$$I \models C \text{ if } \pi_C(I) > 0$$

Now, let us restrict our attention to only those CP-theories in which, for every CP-law r, the sum of the probabilities α_i appearing in $head(r)$ is precisely 1. This

is without loss of generality, since we can simply add an additional disjunct ($P : 1 - \sum_i \alpha_i$), with P some new atom, to all rules which do not satisfy this property. It is easy to see that the set of possible worlds is then independent of the precise values of the α_i, i.e., the qualitative aspects of the semantics of such a theory depend only on the qualitative aspects of its syntactical form. Stated differently, for any pair of CP-theories C, C' which differ only on the α_i's, it holds that, for any interpretation I, $I \models C$ iff $I \models C'$.

From the point of view of disjunctive logic programming, this set of possible worlds therefore offers an alternative semantics for such a program. Under this semantics, the intuitive reading of a rule of form (37) is: "ϕ causes a non-deterministic event, whose effect is precisely one of the $h_1, \ldots, h_n$." This is a different informal reading than in the standard stable model semantics for disjunctive programs (Przymusinski 1991). Indeed, under our reading, a rule corresponds to a causal event, whereas, under the stable model reading, it is supposed to describe an aspect of the reasoning behaviour of a rational agent. Consider, for instance, the disjunctive program $\{p \vee q.\ p.\}$. To us, this program describes a set of two non-deterministic events: One event causes either p or q and another event always causes p. Formally, this leads to two possible worlds, namely $\{p\}$ and $\{p, q\}$. Under the stable model semantics, however, this program states that an agent believes either p or q and the agents believes p. In this case, he has no reason to believe q and the only stable model is $\{p\}$. So, clearly, the causal view on disjunctive logic programming induced by CP-logic is fundamentally different from the standard view and leads to a different formal semantics. Interestingly, the *possible model semantics* (Sakama and Inoue 1994) for disjunctive programs is quite similar to the LPAD treatment, because it consists of the stable models of instances of a program. Because, as shown in Section 8.2, the semantics of CP-logic considers the well-founded models of instances, these two semantics are very closely related. Indeed, for a large class of programs, including all stratified ones, they coincide completely.

Normal logic programs. Let us consider a logic program P, consisting of a set of rules $h \leftarrow \phi$, with h a ground atom and ϕ a formula. Syntactically, such a program is also a deterministic CP-theory. Its semantics π_P assigns a probability of 1 to a single interpretation and 0 to all other interpretations. Moreover, the results from Section 8.2 tell us that the interpretation with probability 1 will be precisely the well-founded model of P. As such, these results show that a logic program under the well-founded semantics can be viewed as a description of deterministic causal information. Concretely, we find that we can read a rule $h \leftarrow \phi$ as: "ϕ causes a deterministic event, whose effect is h."

This observation makes explicit the connection between causal reasoning and logic programming that has long been implicitly present in this field, as is witnessed, e.g., by the work on situation calculus in logic programming. As such, it enhances the theoretical foundations behind the pragmatic use of logic programs to represent causal events.

FO(ID). FO(ID) (also called ID-logic) (Denecker and Ternovska 2007) extends classical logic with inductive definitions. Similar to the way they appear in mathematical texts, an inductive definition is represented as a set of definitional rules, which are of the form $\forall \mathbf{x}\ p(\mathbf{t}) \leftarrow \phi$, where $\mathbf{x}$ is a tuple of variables, ϕ is a first-order formula and $p(\mathbf{t})$ an atom. Such a definition defines all predicates in the head of the rules by simultaneous induction in terms of the other predicates, which are called the *open* predicates of the definition. This syntax offers a uniform way of expressing the most important forms of inductive definitions found in mathematics, including monotone, transfinite and iterated inductive definitions, and inductive definitions over a well-founded order. Formally, the semantics of such a definition is given by the well-founded semantics, which has been shown to correctly formalise these forms of inductive definitions. To be more concrete, an interpretation I is a model of a definition D if it interprets the defined predicates as the well-founded model of D extending the restriction of I to the open symbols of D.

Our results show that finite propositional definitions in FO(ID) are, both syntactically and semantically, identical to deterministic CP-theories. We can therefore view such a set of rules as *both* an inductive definition *and* a description of a causal process. This relation between induction and causality may be remarkable, but it is not all that surprising. In essence, an inductive definition defines a concept by describing how to *construct* it. As such, an inductive definition also specifies a construction process, and such processes are basically causal in nature. Or to put it another way, an inductive definition is nothing more than a description of a causal process, that takes place not in the real world, but in the domain of mathematical objects. This suggests that the ability of mathematicians and formal scientists in general to understand inductive definitions is rooted deeply in human *common sense*, in particular our ability to understand and reason about causation in the physical world.

9 Related work

In this section, we discuss some research that is related to our work on CP-logic. Roughly speaking, we can divide this into two different categories, namely, the related work that focuses mainly on formalizing causality and that which focuses mainly on representing probabilistic knowledge.

9.1 Pearl's causal models

Our work on CP-logic studies causality from a knowledge representation perspective. As such, it is closely related to the work of Pearl (2000). His work uses Bayesian networks and *structural models* as formal tools. In Section 7, we have already compared CP-logic to Bayesian networks and showed that it offers certain representational advantages. A structural model is a set of equations, each of which defines the value of one random variable in terms of the values of a set of other random variables. For each endogenous random variable, there is precisely one such defining equation. As for Bayesian networks, we can say here that a CP-theory is more

detailed, since it represents individual causal laws, while a single structural model equation has to take into account *all* of the random variables that have a direct influence on the value of the defined random variable. Another similarity to Bayesian networks is that structural models have to be acyclic as well, which means that, in this sense, they are also less general than CP-logic. Apart from this, a lot depends of course on the particular form that these equations take, so there is not much more that can be said in general about this.

Pearl's work mainly focuses on the behaviour of causal models in the presence of interventions. As we have shown in Section 7.5, it is possible to consider similar interventions in the context of CP-logic. Pearl uses interventions for a number of interesting purposes, such as handling counterfactuals. They have also been used to define concepts such as "actual causes" (Halpern and Pearl 2001a) and "explanations" (Halpern and Pearl 2001b). The explicitly dynamic processes of CP-logic seem to offer an interesting setting in which to investigate these concepts as well. Indeed, in any particular branch of an execution model of a CP-theory, every true atom p is caused by at least one CP-law whose precondition ϕ was satisfied at the time when this event happened. It now seems sensible to call ϕ an actual cause of p. An interesting question is to what extent such a definition would coincide with the notion of actual causation defined by Halpern. However, we leave these issues for future work.

9.2 Causality in logic programs

In the area of logic programming, many languages can be found which express some kind of (non-probabilistic) causal laws. A typical example is McCain and Turner's *causal theories* (McCain and Turner 1996). A causal theory is a set of rules $\phi \Leftarrow \psi$, where ϕ and ψ are propositional formulas. The semantics of such a theory T is defined by a fixpoint criterion. Concretely, an interpretation I is a model of T if I is the *unique* classical model of the theory T^I that consists of all ϕ, for which there is a rule $\phi \Leftarrow \psi$ in T such that $I \models \psi$.

In CP-logic, we assume that the domain is initially in a certain state, which then changes through a series of events. This naturally leads to the kind of constructive processes that we have used to define the formal semantics of CP-logic. By contrast, according to McCain and Turner's fixpoint condition, a proposition can have any truth value, as long as there exists some causal explanation for this truth value. This difference mainly manifests itself in two ways.

First, in CP-logic, every endogenous property has an initial truth value, which can only change as the result of an event. As such, there is a fundamental asymmetry between falsity and truth, since only one of them represents the "natural" state of the property. For McCain & Turner, however, truth and falsity are completely symmetric and both need to be causally explained. As such, if the theory is to have any models, then, for every proposition Q, there must always be a cause for either Q or $\neg Q$.

A second difference is that the constructive processes of CP-logic rule out any unfounded causality, i.e., it cannot be the case that properties spontaneously cause

themselves. In McCain & Turner's theories, this "spontaneous generation" of properties can occur. For instance, the CP-theory $\{Q \leftarrow Q\}$ has $\{\}$ as its (unique) model, whereas the causal theory $\{Q \Leftarrow Q\}$ has $\{Q\}$ as its (unique) model. As such, the direct representation of cyclic causal relations that is possible in CP-logic (e.g., Example 2) cannot be done in causal theories; instead, one has to use an encoding similar to the one needed in Bayesian networks (e.g., Figure 12). In practice, the main advantage of McCain & Turner's treatment of causal cycles seems to be that it offers a way of introducing exogenous atoms into the language. Indeed, by including both $Q \Leftarrow Q$ and $\neg Q \Leftarrow \neg Q$, one can express that Q can have any truth value, without this requiring any further causal explanation. Of course, CP-logic has no need for such a mechanism, since we make an explicit distinction between exogenous and endogenous predicates. It is interesting to observe that, given the relation between logic programming and causal theories proven in (McCain 1997), this difference actually corresponds to the difference between the well-founded and completion semantics for logic programs.

9.3 Action languages

In Knowledge Representation, a significant amount of work has been on the topic of *action languages*, such as $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ (Gelfond and Lifschitz 1998). These languages have in common with CP-logic that causality plays a significant role in their setting, and also that they are closely related to logic programming (Lifschitz and Turner 1999). Moreover, there also exist probabilistic extensions of these langauges (Baral et al. 2002).

Action languages conceive the world as consisting of a system of causal laws, together with an external agent who can decide to act upon this system. Crucially, the agent's decisions are themselves not governed by the system's causal laws. Therefore, if we consider, e.g., Pearl's framework, the closest thing to this concept of an action would be his notion of an intervention. Indeed, it has been shown in (Tran and Baral 2004) that interventions can be encoded in the probabilistic action language PAL (Baral et al. 2002) precisely as actions. In CP-logic, the most natural way of encoding an action would be, for the same reason, by means of an exogenous predicate. While the behaviour of an agent in an action language is not determined by the state of the system, it *is* however typically restrained by it: certain actions are only available in certain states. Neither of these two ways of encoding actions (i.e., in Pearl's framework or in ours) would be able to directly represent such constraints.

Action languages allow the effects of an action to be specified by means of so-called *dynamic causal laws*. In Pearl's framework, the effect of an action would be given as part of the specification of the intervention that is performed. In CP-logic, such knowledge would take the form of a CP-law whose body contains the exogenous predicate representing the action, and whose head contains some endogenous predicate that is affected by the action.

Besides these dynamic causal laws, there are also *static causal laws*. These represent the causal laws that are obeyed by the system itself; once we know the direct

effects of the agent's actions, the static causal laws tell us how these propagate through the rest of the system. In Pearl's framework, they would correspond to the functional equation that were not intervened with. In CP-logic, they would be CP-laws with endogenous predicates in body and head.

As this brief discussion demonstrates, we could conceive of a probabilistic action language in which the (dynamic and static) causal laws are expressed in CP-logic, while the actions and constraints thereon are defined in some other language. To the best of our knowledge, however, all of the approaches in current literature use essentially a McCain and Turner-style representation for the causal laws (see e.g. (Giunchiglia and Lifschitz 1998)).

To illustrate, let us consider the transmission in a car. This is a system consisting of a number of gear wheels, which can be connected in various ways, such that turning one of the gear wheels causes connected gear wheels to turn also. This system can be represented by a set of causal laws—and can be done so better in CP-logic than in McCain and Turner's logic. The reason is that there exist cyclic causal relations between connected gear wheels: if the engine is turning, this can cause the car to move, but vice versa, if the car is moving, this can also cause the engine to turn ("engine braking"). As explained in Section 9.2, such cyclic causality is handled better in CP-logic. Note also that in the context of probabilistic planning, the McCain and Turner style representation poses the problem that loops in the causal laws (e.g., $p \Leftarrow q$ and $q \Rightarrow p$) lead to uncertainty (e.g., p and q can either both be true or both be false) that is not probabilistically quantified. PAL, for instance, solves this problem by assuming that, in such a case, all possible states are equally likely.

A CP-logic representation of the transmission would probably use predicates such as *Clutch* and *ShiftGear* to refer to the actions available to the driver. These would be exogenous predicates, and we would not say anything more about them. An action language, however, could do more. It would also allow to express constraints on these actions, such as that shifting gears is only allowed while the clutch is in operation. In such a setting, we would have enough information to help the driver come up with a plan to, e.g., put the car into fourth gear without stalling, which cannot be done using just CP-logic by itself.

9.4 Probabilistic languages

Part of the motivation behind CP-logic was to provide a probabilistic logic programming language in which statements have an intuitive meaning that can easily be explained to domain experts, without having to rely on any prior knowledge of logic programming. To this end, we have developed, from scratch, a formalization of the concept of a probabilistic causal law; the resulting language was then shown to be equivalent to the probabilistic logic programming construction of LPADs. This result allows us to interpret probabilistic logic programs in a new way, namely as sets of probabilistic causal laws. This does not only hold for LPADs themselves, but also for related languages, which we will now discuss in more detail.

9.4.1 Independent Choice Logic

Independent Choice Logic (ICL) (Poole 1997) by Poole is a probabilistic extension of abductive logic programming, that extends the earlier formalism of *Probabilistic Horn Abduction* (Poole 1993). An ICL theory consists of both a logical and a probabilistic part. The logical part is an acyclic logic program. The probabilistic part consists of a set of rules of the form (in CP-logic syntax):

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n)$$

such that $\sum_{i=1}^{n} = 1$. The atoms h_i in such clauses are called *abducibles*. Each abducible may only appear once in the probabilistic part of an ICL program; in the logical part of the program, abducibles may only appear in the bodies of clauses.

Syntactically speaking, each ICL theory is also CP-theory. Moreover, the ICL semantics of such a theory (as formulated in, e.g., (Poole 1997)) can easily be seen to coincide with our instance based semantics for LPADs. As such, an ICL theory can be seen as a CP-theory in which every CP-law is either deterministic or unconditional.

We can also translate certain LPADs to ICL in a straightforward way. Concretely, this can be done for acyclic LPADs without exogenous predicates, for which the bodies of all CP-laws are conjunctions of literals. Such a CP-law r of the form:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \phi$$

is then transformed into the set of rules:

$$\left\{ \begin{array}{r} h_1 \leftarrow \phi \wedge Choice_r(1). \\ \cdots \\ h_n \leftarrow \phi \wedge Choice_r(n). \\ (Choice_r(1) : \alpha_1) \vee \cdots \vee (Choice_r(n) : \alpha_n). \end{array} \right\}$$

The idea behind this transformation is that every selection of the original theory C corresponds to precisely one selection of the translation C'. More precisely, if we denote by $ChoiceRule(r)$ the last CP-law in the above translation of a rule r, then a C-selection σ corresponds to the C'-selection σ', for which for all $r \in C$, $\sigma(r) = (h_i : \alpha_i)$ iff $\sigma'(ChoiceRule(r)) = (Choice_r(i) : \alpha_i)$. It is quite obvious that this one-to-one correspondence preserves both the probabilities of selections and the (restrictions to the original vocabulary of the) well-founded models of the instances of selections. This suffices to show that the probability distribution defined by C coincides with the (restriction to the original vocabulary of) the probability distribution defined by C'.

So, our result on the equivalence between LPADs and CP-logic shows that the two parts of an ICL theory can be understood as, respectively, a set of unconditional probabilistic events and a set of deterministic causal events. In this sense, our work offers a causal interpretation for ICL. It is, in this respect, somewhat related to the work of Finzi et al. on causality in ICL. In (Finzi and Lukasiewicz 2003), these authors present a mapping of ICL into Pearl's structural models and use this to derive a concept of actual causation for this logic, based on the work by Halpern

(Halpern and Pearl 2001a). This approach is, however, somewhat opposite to ours. Indeed, we view a CP-theory, with its structure based on individual probabilistic causal laws, as a more fine-grained model of causality. Transforming a CP-theory into a structural model actually loses information, in the sense that it is not possible to recover the original structure of the theory. From the point-of-view of CP-logic, the approach of Finzi et al. would therefore not make much sense, since it would attempt to define the concept of actual causation in a more fine-grained model of causal information by means of a transition to a coarser one.

9.4.2 P-log

P-log (Baral et al. 2004; Baral et al. 2008) is an extension of the language of Answer Set Prolog with new constructs for representing probabilistic information. It is a sorted logic, which allows for the definition of *attributes*, which map tuples (of particular sorts) into a value (of a particular sort). Two kinds of probabilistic statements are considered. The first are called *random selection rules* and are of the form:

$$[r]\ random(A(\mathbf{t}) : \{x : P(x)\}) \leftarrow \phi.$$

Here, r is a name for the rule, P is an unary boolean attribute, A is an attribute with $\mathbf{t}$ a vector of arguments of appropriate sorts, and ϕ is a collection of so-called extended literals[8]. The meaning of a statement of the above form is that, if the body ϕ of the rule is satisfied, the attribute $A(\mathbf{t})$ is selected at random from the intersection of its domain with the set of all terms x for which $P(x)$ holds (unless some deliberate action intervenes). The label r is a name for the experiment that performs this random selection. The choice of which value will be assigned to this attribute is random and, by default, all possible values are considered equally likely. It is, however, possible to override such a default, using the second kind of statements, called *probabilistic atoms*. These are of the form:

$$pr_r(A(\mathbf{t}) = y \mid_c \phi) = \alpha.$$

Such a statement should be read as: if the value of $A(\mathbf{t})$ is determined by the experiment r, and if also ϕ holds, then the probability of $A(\mathbf{t}) = y$ is α.

The information expressed by a random selection rule and its associated probabilistic atoms is somewhat similar to a CP-law, but stays closer to a Bayesian network style representation. Indeed, it expresses that, under certain conditions, the value of a certain attribute will be determined by some implicit random process, which produces each of a number of possible outcomes with a certain probability. We see that, as in Bayesian networks, there is no way of directly representing information about the actual events that might take place; instead, only information about the way in which they eventually affect the value of some attribute (or random variable, in Bayesian network terminology) can be incorporated. Therefore,

[8] An extended literal is either a classical literal or a classical literal preceded by the default negation *not*, where a classical literal is either an atom $A(\mathbf{t}) = t_0$ or the classical negation $\neg A(\mathbf{t}) = t_0$ thereof.

representing the kind of phenomena discussed in Section 7—namely, cyclic causal relations and effects with a number of independent possible causes—requires the same kind of encoding in P-log as in Bayesian networks.

A second interesting difference is that a *random*-statement of P-log represents an experiment in which a value is selected from a *dynamic* set of alternatives, whereas, in CP-logic the set of possible outcomes is specified statically. Consider, for instance, a robot that leaves a room by selecting at random one of the doors that happens to be open. In P-log, this can easily be written down as:

$$[r]\ random(Leave\_through : \{x : Open\_door(x)\}).$$

In CP-logic, such a concise representation is currently not possible.

Apart from probabilistic statements, a P-log program can also contain a set of regular Answer Set Prolog rules and a set of observations and interventions. The difference between observations and interventions is the same as highlighted by Pearl, and (Baral and Hunsaker 2007) shows that interventions in P-log can be used to perform the same kind of counterfactual reasoning as Pearl does. One interesting difference, however, is that in P-log interventions are actually represented *within* the theory, whereas Pearl's approach (as well as the one we presented in Section 7.5) views interventions as meta-manipulations of theories.

In summary, the scope of P-log is significantly broader than that of CP-logic and it is a more full-blown knowledge representation language than CP-logic, which is only aimed at expressing a specific kind of probabilistic causal laws. However, when it comes to representing just this kind of knowledge, CP-logic offers the same advantages over P-log that it does over Bayesian networks.

9.4.3 First-order Versions of Bayesian networks

In this section, we discuss two approaches that aim at lifting the propositional formalism of Bayesian networks to a first-order representation, namely *Bayesian Logic Programs (BLPs)* (Kersting and De Raedt 2000) and *Relational Bayesian Networks (RBNs)* (Jaeger 1997).

A Bayesian Logic Program or BLP consists of a set of definite clauses, using the symbol "|" instead of "←", i.e., clauses of the form

$$P(\mathbf{t}_0) \mid B_1(\mathbf{t_1}), \ldots, B_n(\mathbf{t_n}).$$

in which P and the B_i's are predicate symbols and the $\mathbf{t_j}$'s are tuples of terms. For every predicate symbol P, there is a domain $dom(P)$ of possible values. The meaning of such a program is given by a Bayesian network, whose nodes consist of all the atoms in the least Herbrand model of the program. The domain of a node for a ground atom $P(\mathbf{t})$ is $dom(P)$. For every ground instantiation $P(\mathbf{t}_0) \mid B_1(\mathbf{t_1}), \ldots, B_n(\mathbf{t_n})$ of a clause in the program, the network contains an edge from each $B_i(\mathbf{t_i})$ to $P(\mathbf{t_0})$, and these are the only edges that exist.

To complete the definition of this Bayesian network, all the relevant conditional probabilities also need to be defined. To this end, the user needs to specify, for each clause in the program, a conditional probability table, which defines the conditional

probability of every value in $dom(P)$, given an assignment of values to the atoms in the body of the clause. Now, let us first assume that every ground atom in the Bayesian network is an instantiation of the head of precisely one clause in the program. In this case, the tables for the clauses suffice to determine the conditional probability tables of the network, because every node can then simply take its probability table from this unique clause. However, in general, there might be many such clauses. To also handle this case, the user needs to specify, for each predicate symbol P, a so-called *combination rule*, which is a function that produces a single probability from a multiset of probabilities. The conditional probability table for a ground atom $P(\mathbf{t})$ can then be constructed from the set of all clauses r, such that $P(\mathbf{t})$ is an instantiation of $head(r)$, by finding the appropriate entries in the tables for all such clauses r and then applying the combination rule for P to the multiset of these values. According to the semantics of Bayesian Logic Programs, this combination rule will always be applied, even when there exists only a single such r.

This completes the definition of BLPs as given in, e.g., (Kersting and De Raedt 2000). More recently, a number of issues with this formalism have led to the development of Logical Bayesian Networks (Fierens et al. 2005). These issues have also prompted the addition of so-called "logical atoms" to the original BLP language (Kersting and De Raedt 2007). Since this does not significantly affect any of the comparisons made in this section, however, we will ignore this extension.

A *Relational Bayesian Network* (Jaeger 1997) is a Bayesian network in which the nodes correspond to predicate symbols and the domain of a node for a predicate P/n consists of all possible interpretations of this predicate symbol in some fixed domain D, i.e., all subsets of D^n. The conditional probability distribution associated to such a node P is specified by a *probability formula* F_p. For every tuple $\mathbf{d} \in D^n$, $F_p(\mathbf{d})$ defines the probability of $\mathbf{d}$ belonging to the interpretation of P in terms of probabilities of tuples $\mathbf{d}'$ belonging to the interpretation of a predicate P', where P' is either a parent of P in the graph or even, under certain conditions, P itself. Such a probability formula can contain a number of different operations on probabilities, including the application of arbitrary combination rules. Such a Relational Bayesian Network can also be compiled into a network that is similar to that generated by a BLP, i.e., one in which the nodes correspond to domain atoms instead of predicate symbols. The main advantage of such a compiled network is that it allows more efficient inference.

Again, the main difference between these two formalisms and CP-logic is that they both stick to the Bayesian network style of modeling, in the sense that the actual events that determine the values of the random variables are entirely abstracted away and only the resulting conditional probabilities are retained. However, through the use of, respectively, combination rules and probability formulas, these can be represented in a more structured manner than in a simple table. In this way, knowledge about, the underlying causal events can be exploited to represent the conditional probability distributions in a concise way. The most common example is probably the use of the *noisy-or* to handle an effect which has a number of independent possible causes. For instance, let us consider the Russian roulette

problem of Example 15. In a BLP, the relation between the guns firing and the player's death could be represented by the following clause:

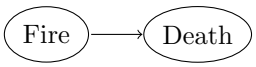
$$Death \mid Fire(X).$$

	$Fire(x) = \mathbf{t}$	$Fire(x) = \mathbf{f}$
$Death = \mathbf{t}$	1/6	0
$Death = \mathbf{f}$	5/6	1

Combination rule for $Death$: *noisy-or*

In Relational Bayesian Networks, this would be represented as follows:

$$F_{Death} = \textit{noisy-or}(\{1/6 \cdot Fire(x) \mid x\})$$



As such, combination rules do allow some knowledge about the events underlying the conditional probabilities to be incorporated into the model. However, this is of course not the same as actually having a structured representation of the events themselves, as is offered by CP-logic. As a consequence of this, cyclic causal relations, such as that of our *Pneumonia-Angina* example, still need the same kind of encoding as in a Bayesian network.

9.4.4 Other approaches

In this section, we give a quick overview of some other related languages. An important class of probabilistic logic programming formalisms are those following the *Knowledge Based Model Construction* approach. Such formalisms allow the representation of an entire "class" of propositional models, from which, for a specific query, an appropriate model can then be constructed "at run-time". This approach was initiated by Breese (Breese 1992) and Bacchus (Bacchus 1993) and is followed by both Bayesian Logic Programs and Relational Bayesian Networks. Other formalism in this class are *Probabilistic Knowledge Bases* of Ngo and Haddawy (Ngo and Haddawy 1997) and *Probabilistic Relational Models* of Getoor et al. (Getoor et al. 2001). From the point of view of comparison to CP-logic, both are very similar to Bayesian Logic Programs (see, e.g., (Kersting and De Raedt 2001) for a comparison).

The language used in the *Programming in Statistical Modeling* system (PRISM) (Sato and Kameya 1997) is very similar to Independent Choice Logic. Our comments concerning the relation between CP-logic and Independent Choice Logic therefore carry over to PRISM.

Like CP-logic, *Many-Valued Disjunctive Logic Programs* (Lukasiewicz 2001) are also related to disjunctive logic programming. However, in this language, probabilities are associated with disjunctive clauses as a whole. In this way, uncertainty of the implication itself—and not, as is the case with LPADs or CP-logic, of the disjuncts in the head—is expressed.

All the works mentioned so far use point probabilities. There are however also a number of formalisms using probability intervals: *Probabilistic Logic Programs* of Ng and Subrahmanian (Ng and Subrahmanian 1992), their extension to *Hybrid Probabilistic Programs* of Dekhtyar and Subrahmanian (Dekhtyar and Subrahmanian 2000) and *Probabilistic Deductive Databases* of Lakshmanan and Sadri (Lakshmanan and Sadri 1994). Contrary to our approach, programs in these formalisms do not define a *single* probability distribution, but rather a *set* of possible probability distributions, which allows one to express a kind of "meta-uncertainty", i.e., uncertainty about which probability distribution is the "right" one. Moreover, the techniques used by these formalisms tend to have more in common with constraint logic programming than standard logic programming. The more recent formalism of CLP(BN) (Santos Costa et al. 2003) belongs to this class.

We also want to mention *Stochastic Logic Programs* of Muggleton and Cussens (Cussens 2000; Muggleton 2000), which is a probabilistic extension of Prolog. In this formalism, probabilities are attached to the selection of clauses in Prolog's SLD-resolution algorithm, which basically results in a first-order version of stochastic context free grammars. Because of this formalism's strong ties to the procedural aspects of Prolog, it appears to be quite different from CP-logic and indeed all of the other formalisms mentioned here.

ProbLog (De Raedt et al. 2007) is a more recent probabilistic extension of pure Prolog. Here, too, every clause is labeled with a probability. The semantics of ProbLog is very similar to that of LPADs and, in fact, the semantics of a *ground* ProbLog program coincides completely with that of the corresponding LPAD. More precisely put, a ProbLog rule of the form:

$$\alpha : \quad h \leftarrow b_1, \ldots, b_n,$$

where h and the b_i are ground atoms is entirely equivalent to the LPAD rule:

$$(h : \alpha) \leftarrow b_1, \ldots, b_n.$$

For non-ground programs, however, there is a difference. The semantics of an LPAD first grounds the entire program and then probabilistically selects instantiations of the rules of this ground program. In ProbLog, on the other hand, selections directly pick out rules of the original program. This means that, for instance, the following ProbLog-rule:

$$0.8 : \quad likes(X, Y) \leftarrow likes(X, Z), likes(Z, Y),$$

specifies that, with probability 0.8, the *likes*-relation is entirely transitive, whereas the corresponding LPAD-rule would mean that for all *individuals* a, b and c, the fact that a likes b and b likes c causes a to like c with probability 0.8.

10 Conclusions and future work

Causality has an inherent dynamic aspect, which can be captured at the semantical level by the probability tree framework that Shafer has developed in (Shafer 1996). He concludes this book with the following observation:

When we think of a Bayes net as a representation of a probability tree, we sometimes may also want to leave indeterminate orderings that are not imposed by arrows in the graph, so that the net can be thought of as a representation not of a single tree but of a class of trees, corresponding to different choices for these orderings. The possibility of introducing indeterminacy in the ordering of judgements is obviously equally present in the type-theoretical representation. [...] [W]e can think of [a large collection of partially ordered judgements] as a set of rules from which martingale trees can be constructed. [...] More abstractly, they can be thought of as causal laws, and we can imagine many problems of deliberation being posed and solved directly in terms of these causal laws, without the specification of a martingale tree. Thus type theory can take us beyond probability trees to a more general framework for causal deliberation.

This paper can be seen as an extension of Shafer's work in the direction pointed at by the above remarks. We have developed a logical language which uses *probabilistic causal laws* to concisely represent classes of probability trees. Our representation does not explicitly impose any order on the possible events, since we move away from a representation in which it is the outcome of previous events that causes a new event, to one in which new events are caused by properties of the current state of the domain. Even though this means that the probability trees corresponding to a given set of causal laws might be considerably different from one another, we prove that they will all still generate the same probability distribution over their final states. Therefore, such causal laws capture precisely those properties of probability trees that we need to answer questions about the probabilities in these final states, which is what we are typically interested in.

A first contribution of this paper is the language CP-logic itself. This language allows to represent a probability distribution over possible states of a domain by an enumeration of the probabilistic causal laws according to which it is generated. As we tried to show by, among others, the comparison to Bayesian networks, this representation is often natural and concise. A second contribution is that we have shown that CP-logic can be equivalently defined as a probabilistic logic programming language. Because both the meaning of statements in CP-logic, as well as their formal semantics, can be completely explained in terms of intuitions about probabilistic causal laws, this formal equivalence offers a new way of (informally) explaining the meaning of probabilistic logic programs. This is a useful contribution to the existing modeling methodology for such languages.

By relating causality and logic programming in this way, our paper also serves as a unifying semantic study of existing probabilistic and non-probabilistics logics and formalisms. We showed how CP-logic refines causal Bayesian networks and several logics based on them. We also elaborated on the links between CP-logic and existing logic programming extensions such as ICL, PRISM and LPADs, thus showing that these logics can also be viewed as causal probabilistic logics. For example, a theory in ICL can be understood as a combination of deterministic causal events and unconditional probabilistic events. As for logic programming itself, we showed that CP-logic induces a causal view on this formalism, in which rules represent deterministic causal events. We also argued that this view basically coincides with the view of logic programs as inductive definitions. To be more concrete, we have shown that a normal logic program under the well-founded semantics can be understood

as a set of deterministic causal statements and we have presented an alternative semantics for disjunctive logic programs (similar to that of (Sakama and Inoue 1994)) under which these can be interpreted as sets of non-deterministic causal events.

This paper is primarily intended to show how the concept of a probabilistic causal law can be formalized in a logical language, and to demonstrate the close relation of such a language to probabilistic logic programs. Because of this, we have intentionally kept our language quite simple. As became apparent in the comparison with other logics, such as P-log, CP-logic therefore lacks the expressivity to be truly useful for a broad class of applications. To make it more suitable for practical purposes, it should therefore be improved in a number of ways. We see the following opportunities for future research.

Refinement of CP-logic. The current language of CP-logic is restricted in a number of ways. First, it only allows a finite number of CP-laws. Let us consider, for instance, a die that is rolled as long as it takes to obtain a six. Here, there is no upper bound on the number of throws that might be needed and, therefore, this example can currently not be represented in CP-logic. Second, CP-logic is also limited in its representation of the effects of an event. For instance, it is not possible to directly represent events whose range of possible outcomes is not completely fixed beforehand. Also, we currently do not allow different events to cancel out or reinforce each other's effects. Third, and somewhat related to the previous point, CP-logic currently can only handle properties that are either fully present or fully absent. As such, it cannot correctly represent causes which have only a contributory effect, e.g., turning on a tap would not instantaneously cause a basin to be full, but only contribute a certain amount per time unit.

Integration into a larger formalism. To correctly formalise a domain in CP-logic, a user must exactly know the causes and effects of all relevant events that might happen. For real domains of any significant size, this is an unrealistic assumption. Indeed, typically, one will only have such detailed knowledge about certain parts of a domain. So, in order to still be able to use CP-logic in such a setting, it would have to be integrated with other forms of knowledge. There are some obvious candidates for this: statements about the probabilities of certain properties, statements about probabilistic independencies (such as those in Bayesian networks), and constraints on the possible states of the domain. Integrating these different forms of knowledge without losing conceptual clarity is one of the main challenges for future work regarding CP-logic, and perhaps even for the area of uncertainty in artificial intelligence as a whole.

Inference. The most obvious inference task in the context of CP-logic is calculating the probability $\pi_C(\phi)$ of a formula ϕ. A straightforward way of doing this would be to exploit the relation between CP-logic and (probabilistic) logic programming, such that we perform these computations by reusing existing algorithms (e.g., the inference algorithm of Poole's independent choice logic (Poole 1997)) in an appropriate way. A more advanced technique, using binary decision diagrams, has recently been

developed in (Riguzzi 2007). Another interesting inference task concerns the construction of a theory in CP-logic. For probabilistic modeling languages in general, it is typically not desirable that a user is forced to estimate or compute concrete probability values herself; instead, it should be possible to automatically derive these from a given data set. For CP-logic, there already exist algorithms that are able to do this in certain restricted cases (Riguzzi 2004; Blockeel and Meert 2007). It would be interesting to generalize these, in order to make them generally applicable. Besides such learning of probabilistic parameters, it is also possible to learn the structure of the theory itself. This too is an important topic, because if we are able to construct the theory that best describes a given data set, we are in effect finding out which causal mechanisms are most likely present in this data. Such information can be relevant for many domains. For instance, when bio-informatics attempts to distinguish active from non-active compounds, this is exactly the kind of information that is needed. In (Meert et al. 2007), it is discussed how certain Bayesian network learning techniques can be adapted to perform structure learning for ground CP-logic.

Acknowledgements

This research was supported by GOA 2003/8 Inductive Knowledge Bases and by FWO Vlaanderen. Joost Vennekens is a postdoctoral researcher of the FWO.

Appendix A Proofs of the theorems

In this section, we present proofs of the theorems that were stated in the previous section. To ease notation, we will assume that there are no exogenous predicates. This can be done without loss of generality, since all our results can simply be relativized with respect to some fixed interpretation for these predicates.

A.1 The semantics is well-defined

We start by proving that the semantics of CP-logic—and in particular, the partial interpretation ν_s, the potential in s, used in the additional condition imposed by Definition 11 for handling negation—is indeed well-defined. Since we defined ν_s as the unique limit of all terminal hypothetical derivation sequences of s, this requires us to show that all such sequences indeed end up in the same limit (Theorem 4).

Let us consider a CP-theory C and state s in an execution model of C. We will denote by $\mathcal{R}(s)$ the set of all CP-laws $r \in C$ that have not yet happened in s, i.e., for which there is no ancestor s' of s with $\mathcal{E}(s') = r$. Consider the collection O_s of all partial interpretions ν such that for each atom p, $p^\nu = \mathbf{t}$ iff $p^{\mathcal{I}(s)} = \mathbf{t}$, and for each rule $r \in \mathcal{R}(s)$, if $body(r)^\nu \neq \mathbf{f}$, then for each atom $p \in head_{At}(r)$, $p^\nu \neq \mathbf{f}$. Stated differently, ν can be obtained from $\mathcal{I}(s)$ by turning false atoms of $\mathcal{I}(s)$ into unknown atoms in such a way that if the body of some rule $r \in \mathcal{R}(s)$ is unknown or true in ν, then each of its head atoms is unknown or true in ν as well.

Proposition 1
Let $(\nu_i)_{0 \leq i \leq n}$ be a hypothetical derivation sequence in state s.

- For each $0 \leq i \leq n$ and each $\nu \in O_s$ it holds that $\nu \leq_p \nu_i$.
- The limit $\nu_n = \nu_s$ is an element of O_s.

Proof
The first property can be proven by a straigthforward induction. Clearly, it holds that $\nu \leq_p \nu_0 = \mathcal{I}(s)$. Assume $\nu \leq_p \nu_i$ for some $i < n$. The true atoms of ν and ν_{i+1} are those of $\mathcal{I}(s)$, so they are the same. Therefore, it suffices to show that every atom p that is false in ν is also false in ν_{i+1}, or, since ν and ν_{i+1} have the same true atoms, that every such p is not unknown in ν_{i+1}. Assume towards contradiction that p is false in ν and unknown in ν_{i+1}. By the induction hypothesis, p is still false in ν_i. Therefore, p belongs to the head of some rule $r \in \mathcal{R}(s)$ such that $body(r)^{\nu_i} \neq \mathbf{f}$. Since $\nu \leq_p \nu_i$, this would imply that $body(r)^{\nu} \neq \mathbf{f}$, which, given that $\nu \in O_s$, leads to the contradiction that $p^{\nu} \neq \mathbf{f}$. Hence, p is false in ν_{i+1}. It follows that $\nu \leq_p \nu_{i+1}$.

As for the second property, it is clear that ν_s can be obtained from $\mathcal{I}(s)$ by turning some false atoms into unknown atoms, and that there are no more rules $r \in \mathcal{R}(s)$ with a non-false body and false atoms in the head w.r.t. ν_s. Hence, $\nu_s \in O_s$. □

We can now use this set O_s to characterize the limit ν_n of any hypothetical derivation sequence $(\nu_i)_{0 \leq i \leq n}$ in s.

Theorem 10
Let $(\nu_i)_{0 \leq i \leq n}$ be a hypothetical derivation sequence in s and let ν be the least upperbound of O_s w.r.t. the precision order $\leq_p$. Then $\nu_n = \nu$.

Proof
It is obvious that ν itself also belongs to O_s. Therefore, by the first bullet of Proposition 1, $\nu \leq_p \nu_n$. Because, by the second bullet of Proposition 1, ν_n also belongs to O_s, we have that $\nu \geq_p \nu_n$ as well. □

Since this theorem shows that all hypothetical derivation sequences converge to the most precise element of O_s, it implies Theorem 4 and, therefore, our semantics is indeed well defined.

A.2 CP-logic and LPADs are equivalent

Let C be an LPAD. Let us define a *partial C-selection* as a partial function σ from C mapping rules r of a subset $dom(\sigma) \subseteq C$ to pairs $(p : \alpha) \in head^*(r)$. The probability function of selections can be extended to partial selections by setting $P(\sigma) = \prod_{r \in dom(\sigma)} \sigma^{\alpha}(r)$. Define also $S(\sigma)$ as the set of C-selections that extend σ. The following equation is obvious:

$$P(\sigma) = \sum_{\sigma' \in S(\sigma)} P(\sigma')$$

We define an *instance* of σ as any instance $C^{\sigma'}$ in which σ' is a C-selection that extends σ.

Let $\mathcal{T}$ be an execution model of C. Clearly, each node s in $\mathcal{T}$ determines a unique partial C-selection, denoted $\sigma(s)$. Formally, if $(s_i)_{0\leq i\leq n}$ is the path from the root to s, then the domain of $\sigma(s)$ is $\{\mathcal{E}(s_i) \mid 0 \leq i < n\}$ and each rule $r = \mathcal{E}(s_i)$ in its domain is mapped to the atom $p \in head^*(r)$ that was selected for s_{i+1}. Moreover, we have

$$\mathcal{P}(s) = P(\sigma(s)) = \sum_{\sigma' \in S(\sigma(s))} P(\sigma'). \tag{A1}$$

With the path $(s_i)_{0\leq i\leq n}$ from the root to some node s, we now also associate a sequence of partial interpretations $(K_j)_{j=0}^{2n+1}$ defined as follows:

- $K_0 = \bot$, the partial interpretation mapping all atoms to $\mathbf{u}$.
- $K_{2i+1} = \nu_{s_i}$, for all $0 \leq i \leq n$.
- $K_{2i+2} = \nu_{s_i}[p : \mathbf{t}]$, for all $0 \leq i < n$, where p is the head atom of $\mathcal{E}(s_i)$ selected to obtain s_{i+1}.

Proposition 2
For each $\sigma \in \sigma(s)$, $(K_j)_{j=0}^{2n+1}$ is a well-founded induction of C^σ.

Proof
The proof is by induction on the length n of the path from the root of $\mathcal{T}$ to s.

We start by proving that $(K_j)_{j=0}^{2n}$ is a well-founded induction of all instances C^σ with $\sigma \in S(\sigma(s))$. If $n = 0$, then s is the root of the tree and $\sigma(s)$ is the empty partial selection. The sequence (K_0) is obviously a well-founded induction of any instance C^σ. For $n > 0$, the induction hypothesis states that $(K_j)_{j=0}^{2n-1}$ is a well-founded induction of all instances C^σ, where σ belongs to $S(\sigma(s_{n-1}))$. Let r be $\mathcal{E}(s_{n-1})$, the rule selected in s_{n-1}, and let $K_{2n} = K_{2n-1}[p : \mathbf{t}]$ where p was selected in the head of r to obtain s. Hence, $body(r)$ is true in $K_{2n-1} = \nu_{s_{n-1}}$. Clearly, for each $\sigma \in S(\sigma(s))$, C^σ contains the rule $p \leftarrow body(r)$. Consequently, $(K_j)_{j=0}^{2n}$ is a well-founded induction of C^σ.

Next, we prove that $(K_j)_{j=0}^{2n+1}$ is a well-founded induction of all C^σ with $\sigma \in S(\sigma(s))$. Let us investigate the set U of all atoms q such that $K_{2n}(q) \neq K_{2n+1}(q)$. We will prove that all atoms of U are unknown in K_{2n} and false in K_{2n+1} and that U is an unfounded set of C^σ. It then will follow that $(K_j)_{j=0}^{2n+1}$ is a well-founded induction of C^σ.

Let us first verify that all atoms in U are unknown in K_{2n} and false in K_{2n+1}. If $n = 0$, then $K_0 = \nu_0 = K_1$, so $U = \{\}$ and the statement trivially holds. Let $n > 0$. Recall that K_{2n} is $\nu_{s_{n-1}}[p : \mathbf{t}]$, where p is the atom selected in the head of $\mathcal{E}(s_{n-1})$ to obtain s, and $K_{2n+1} = \nu_s$. It is easy to see that the true atoms of K_{2n} and K_{2n+1} are identical to those true in $\mathcal{I}(s)$. Hence, K_{2n} and K_{2n+1} only differ on false or unknown atoms. To show that U contains only atoms that are unknown in K_{2n} and false in K_{2n+1}, it therefore suffices to show that all atoms false in K_{2n} are also false in K_{2n+1}. To prove this, it suffices to show that $K_{2n} \in O_s$. Indeed, if $K_{2n} \in O_s$, Proposition 1 entails that $\nu_s \geq_p K_{2n}$ and hence, all atoms false in K_{2n} are false in $\nu_s = K_{2n+1}$.

We observe that, since $\nu_{s_{n-1}}$ belongs to $O_{s_{n-1}}$ (Proposition 1), all head atoms of rules $r \in \mathcal{R}(s_{n-1})$ with a non-false body in $\nu_{s_{n-1}}$, are true or unknown in $\nu_{s_{n-1}}$. In particular, $\mathcal{E}(s_{n-1}) \in \mathcal{R}(s_{n-1})$ and has a true body in $\nu_{s_{n-1}}$, hence p is true or unknown in $\nu_{s_{n-1}}$. It follows that:

$$\nu_{s_{n-1}} \leq_p \nu_{s_{n-1}}[p : \mathbf{t}] = K_{2n}.$$

It follows that any rule $r \in \mathcal{R}(s) \subseteq \mathcal{R}(s_{n-1})$ with a non-false body in K_{2n}, has a non-false body in $\nu_{s_{n-1}}$; hence, all atoms in the head of such an r are true or unknown in $\nu_{s_{n-1}}$ and, *a fortiori*, in $K_{2n} = \nu_{s_{n-1}}[p : \mathbf{t}]$. Thus, we obtain that $K_{2n} \in O_s$, as desired.

So far, we have proven that $K_{2n+1} = K_{2n}[U : \mathbf{f}]$ and that all elements in U are unknown in K_{2n}. It follows that $K_{2n} \leq_p K_{2n+1}$ and, more generally, that $K_j \leq_p K_{2n+1}$, for all $j \leq 2n$. All that remains to be shown is that U is an unfounded set of each instance of $\sigma(s)$. Let C' be such an instance and for any atom $q \in U$, let $q \leftarrow \varphi$ be a rule of C'. We need to show that φ is false in K_{2n+1}. The rule is obtained as an instance of some rule $r \in C$ with q in its head. The rule r is not one of the rules $\mathcal{E}(s_i)$ with $i < n$, since otherwise q would be true in $\mathcal{I}(s_j)$ for all $j > i$ and, in particular, also in $\nu_{s_n} = K_{2n+1}$, which would contradict the fact that we have already shown q to be false in K_{2n+1}. It follows that $r \in \mathcal{R}(s)$. Since $K_{2n+1} = \nu_s \in O_s$ and q is false in ν_s, $body(r) = \varphi$ is false in K_{2n+1}.

□

Proposition 3
For each leaf l of an execution model $\mathcal{T}$ of C, $\mathcal{I}(l)$ is the well-founded model of each instance C^σ with $\sigma \in S(\sigma(l))$.

Proof
Let l be a leaf and $\sigma \in S(\sigma(l))$. By Proposition 2, $(K_j)_{j=0}^{2n+1}$ is a well-founded induction of C^σ. Because l is a leaf, we have that for every rule $r \in \mathcal{R}(l)$, $body(r)$ is false in $\mathcal{I}(l)$. Therefore, $\mathcal{I}(l) \in O_l$ and Proposition 1 states that $\nu_l \geq_p \mathcal{I}(l)$. However, because $\mathcal{I}(l)$ is two-valued, this implies that $\nu_l = \mathcal{I}_l$. Therefore, $K_{2n+1} = \nu_l$ is a total interpretation. Because a well-founded induction with a total limit is terminal, $\mathcal{I}(l)$ is the well-founded model of C^σ. □

This now allows us to prove the desired equivalence, which was previously stated as Theorem 9.

Theorem 11
Let $\mathcal{T}$ be an execution model of a CP-theory C. For each interpretation J,

$$\mu_C(J) = \pi_{\mathcal{T}}(J).$$

Proof
Given an execution model $\mathcal{T}$ of a CP-theory C (Def. 11), we associate to each node s of $\mathcal{T}$ the set $S(\sigma(s))$ of all those C-selections σ (Def. 17) that extend $\sigma(s)$. It is easy to see that, with $L_{\mathcal{T}}$ the set of all leaves of $\mathcal{T}$, the class $\{S(\sigma(l))|l \in L_{\mathcal{T}}\}$

is a partition of the set $\mathcal{S}_C$ of all selections. Let $L_{\mathcal{T}}(J)$ be the set of all leaves l of $\mathcal{T}$ for which $\mathcal{I}(l) = J$, and let $Sels(J)$ be the set of selections σ such that $WFM(C^\sigma) = J$. Because for each leaf l, the well-founded model of a selection $\sigma \in S(\sigma(l))$ for is $\mathcal{I}(l)$ (Proposition 3), the class $\{S(\sigma(l))|l \in L_{\mathcal{T}}(J)\}$ is a partition of the collection $Sels(J)$. This now allows us to derive the following equation:

$$\begin{array}{lll} \mu_C(J) = \sum_{\sigma \in Sels(J)} P(\sigma) & = \sum_{l \in L_{\mathcal{T}}(J)} \sum_{\sigma \in S(\sigma(l))} P(\sigma) & \text{(Def. 19)} \\ & = \sum_{l \in L_{\mathcal{T}}(J)} \mathcal{P}(l) & \text{(see equation (A1))} \\ & = \pi_{\mathcal{T}}(J). & \end{array}$$

□

For any execution model $\mathcal{T}$ of C, this theorem now characterizes the probability distribution $\pi_{\mathcal{T}}$ in a way that depends only on C and not on $\mathcal{T}$ itself. It follows that, indeed, for all execution models $\mathcal{T}$ and $\mathcal{T}'$ of C, $\pi_{\mathcal{T}} = \pi_{\mathcal{T}'}$, which means that we have now also proven Theorem 6 (and, therefore, Theorem 2 as well).

A.3 Execution models that follow the timing

In this section, we will prove Theorem 7, which states that every stratified CP-theory has an execution model which follows its stratification. Recall that a CP-theory is stratified if it strictly respects some timing λ (i.e., for all $h \in head_{At}(r)$ and $b \in body^+_{At}(r)$, $\lambda(h) \geq \lambda(b)$ and for all $h \in head_{At}(r)$ and $b \in body^-_{At}(r)$, $\lambda(h) > \lambda(b)$). As we did in Definition 9 of Section 4, we will again introduce a event-timing κ of λ (i.e., κ maps rules to time points in such a way that $\lambda(h) \geq \kappa(r) \geq \lambda(b)$ for all $h \in head_{At}(r)$ and $b \in body_{At}(r)$). Moreover, we assume that κ is such that for all $b \in body^-_{At}(r)$, $\kappa(r) > \lambda(b)$. It can easily be seen that for any stratified theory C, it is always possible to find such a κ.

Our goal is now to show that, first, all weak execution models that follow κ (Def. 9) also satisfy temporal precedence and, second, that such a process indeed exists.

Let us start by making some general observations about any weak execution model $\mathcal{T}$ that follows κ. For any descendant s' of a node s of $\mathcal{T}$, it is, by definition, the case that $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$. Because every event r can only affect the truth value of atoms with timing $\geq \kappa(r)$, it must be the case that, for each rule r with timing $< \kappa(\mathcal{E}(s))$, $body(r)^{\mathcal{I}(s)} = body(r)^{\mathcal{I}(s')}$. Suppose now that for such an r it would be the case that $r \in \mathcal{R}(s)$ and $\mathcal{I}(s) \models body(r)$, i.e., r is an event that could also have happened in s. In this case, $body(r)$ would remain satisfied in all descendants of s, up to and including each leaf l that might be reached. However, it is impossible that r actually happens in some descendant s' of s, since that would violate the constraint that $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$. So, it would be the case that $r \in \mathcal{R}(l)$ and $\mathcal{I}(l) \models body(r)$, which would contradict the fact that l is a leaf. We conclude that such an r cannot exist, i.e., for each s, it must be the case that $\mathcal{E}(s)$ is a rule with minimal timing among all rules $r \in \mathcal{R}(s)$ for which $\mathcal{I}(s) \models body(r)$.

Let us now assume that we non-deterministically construct a probabilistic Σ-process $\mathcal{T}$ as follows:

- We start with only a root s, with $\mathcal{I}(s) = \{\}$;

- As long as one exists, we select a leaf s of our current tree, for which the set of rules $r \in \mathcal{R}(s)$ such that $\mathcal{I}(s) \models body(r)$ is non-empty. We then extend $\mathcal{T}$ by executing one of the rules whose timing is minimal in this set.

As shown in the previous paragraph, all weak execution models that follow κ can be constructed in this way. Conversely, each process $\mathcal{T}$ that we can construct in this way can easily be seen to also be a weak execution model. Moreover, it is again easy to see that for all descendants s' of s of such a $\mathcal{T}$ and each rule r with timing $< \kappa(\mathcal{E}(s))$, $body(r)^{\mathcal{I}(s)} = body(r)^{\mathcal{I}(s')}$. Therefore, as we go along any particular branch of $\mathcal{T}$, the minimum timing of all rules with true body can only increase, which means that each process constructed in the above way must follow κ. So, this provides an alternative, constructive characterization of the set of all weak execution models that follow κ. An immediate consequence is that there exist such processes. Therefore, it now suffices to show that all these processes also satisfy temporal precedence.

Proposition 4
Each weak execution model $\mathcal{T}$ that follows the timing κ also satisfies temporal precedence and is, therefore, an execution model.

Proof
We need to show that, for each node s of $\mathcal{T}$, $\nu_s(body(\mathcal{E}(s)) = \mathbf{t}$. In general, applying an event with timing $\geq i$ during a hypothetical derivation sequence only modifies atoms with timing $\geq i$, and hence, can only modify the truth value of bodies of events with timing $\geq i$. Because, in the first step ν_0 of a sequence constructing ν_s, the only events that can be used are those $r \in \mathcal{R}(s)$ for which $\mathcal{I}(s) \models body(r)$ and we know that the time of $\mathcal{E}(s)$ is minimal among these events, we conclude that $\mathcal{I}(s)$ and ν_s coincide on all atoms p with timing $\lambda(p) < i$. Because C strictly respects λ, all atoms $p \in body^-_{At}(r)$ therefore have the same truth value in ν_s as in $\mathcal{I}(s)$. Moreover, $\mathcal{I}(s) \leq_t \nu_s$, so, in particular, for all atoms $p \in body^+_{At}(r)$, $\mathcal{I}(s)(p) \leq_t \nu_s(p)$. By a well-known monotonicity property of three-valued logic, $\mathbf{t} = body(r)^{\mathcal{I}(s)} \leq_t body(r)^{\nu_s}$. Hence, $body(r)^{\nu_s}$ is indeed $\mathbf{t}$. □

This concludes our proof of Theorem 7. Since this theorem clearly generalizes Theorem 3, we have now proven all theorems stated in this paper.

References

Bacchus, F. 1993. Using first-order probability logic for the construction of Bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, UAI'93*. 219–226.

Baral, C., Gelfond, M., and Rushton, N. 2004. Probabilistic reasoning with answer sets. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*. Lecture notes in artifical intelligence (LNAI), vol. 2923. Springer-Verlag, 21–33.

Baral, C., Gelfond, M., and Rushton, N. 2008. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*. to appear.

Baral, C. and Hunsaker, M. 2007. Using the probabilistic logic programming language P-log for causal and counterfactual reasoning and non-naive conditioning. In *Proceedings of IJCAI*.

Baral, C., Tran, N., and Tuan, L. 2002. Reasoning about actions in a probabilistic setting. In *AAAI*.

Blockeel, H. and Meert, W. 2007. Towards learning non-recursive LPADs by transforming them into Bayesian networks. In *Inductive Logic Programming, ILP'06, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 4455. 94–108.

Breese, J. 1992. Construction of belief and decision networks. *Computational intelligence 8,* 4, 624–647.

Comley, J. W. and Dowe, D. L. 2003. General Bayesian networks and asymmetric languages. In *Proceedings of the Second Hawaii International Conference on Statistics and Related fields*.

Cussens, J. 2000. Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 115–122.

De Finetti, B. 1937. La prvision: ses lois logiques, ses sources subjectives. *Annales de l'Institut Henri Poincaré 7*, 1–68.

De Raedt, L., Kimmig, A., and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*. 2462–2467.

Dekhtyar, A. and Subrahmanian, V. S. 2000. Hybrid probabilistic programs. *Journal of Logic Programming 43,* 3, 187–250.

Denecker, M. 1998. The well-founded semantics is the principle of inductive definition. In *Logics in Artificial Intelligence (JELIA'98)*, J. Dix, L. Fariñas del Cerro, and U. Furbach, Eds. Lecture Notes in Artificial Intelligence, vol. 1489. Springer-Verlag, 1–16.

Denecker, M. and Ternovska, E. 2007. A logic of non-monotone inductive definitions. *Transactions On Computational Logic (TOCL)*.

Denecker, M. and Vennekens, J. 2007. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Proceedings*. Lecture Notes in Artificial Intelligence, vol. 4483. Springer, 84–96.

Fierens, D., Blockeel, H., Bruynooghe, M., and Ramon, J. 2005. Logical Bayesian networks and their relation to other probabilistic logical models. In *Proceedings of the 15th International Conference on Inductive Logic Programming, ILP'05*. Lecture Notes in Computer Science, vol. 3625. Springer, 121–135.

Finzi, A. and Lukasiewicz, T. 2003. Structure-based causes and explanations in the independent choice logic. In *Proceedings of the 19h Conference on Uncertainty in Artificial Intelligence, UAI'03*.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–387.

Gelfond, M. and Lifschitz, V. 1993. Representing Action and Change by Logic Programs. *Journal of Logic Programming 17,* 2,3,4, 301–322.

Gelfond, M. and Lifschitz, V. 1998. Action languages. *Linköping Electronic Articles in Computer and Information Science 3,* 16.

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. 2001. Learning probabilistic relational models. In *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds. Springer-Verlag, 7–34.

Ghahramani, Z. 1998. Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 168–197.

Giunchiglia, E. and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of AAAI 98*.

Halpern, J. and Pearl, J. 2001a. Causes and explanations: A structural model approach – part I: Causes. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, UAI'01*.

Halpern, J. and Pearl, J. 2001b. Causes and explanations: A structural model approach – part II: Explanations. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, UAI'01*.

Jaeger, M. 1997. Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*.

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

Kersting, K. and De Raedt, L. 2000. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, J. Cussens and A. Frisch, Eds. 138–155.

Kersting, K. and De Raedt, L. 2001. Bayesian logic programs. Tech. Rep. 151, Institute for Computer Science, University of Freiburg, Germany.

Kersting, K. and De Raedt, L. 2007. Bayesian logic programming: Theory and tool. In *An Introduction to Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press. To appear.

Lakshmanan, L., V. S. and Sadri, F. 1994. Probabilistic deductive databases. In *Proceedings of the International Symposium on Logic Programming, ILPS'94*, M. Bruynooghe, Ed. MIT Press, 254–268.

Lifschitz, V. and Turner, H. 1999. Representing transition systems by logic programs. In *LPNMR*.

Lukasiewicz, T. 2001. Fixpoint characterizations for many-valued disjunctive logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*. Lecture Notes in Artificial Intelligence, vol. 2173. Springer-Verlag, 336–350.

Martin-Löf, P. 1982. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress of Logic, Methodology, and Philosophy of Science*. 153–175.

McCain, N. 1997. Causality in commonsense reasoning about actions. Ph.D. thesis, University of Texas at Austin.

McCain, N. and Turner, H. 1996. Causal theories of action and change. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference (13th AAAI/8th IAAI)*. AAAI Press, 460–465.

Meert, W., Struyf, J., and Blockeel, H. 2007. Learning ground CP-logic theories by means by Bayesian network techniques. In *Proceedings of the 6th International Workshop on Multi-Relational Data Mining*. 93–104.

Muggleton, S. 2000. Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence 5,* 041, 141–153.

Ng, R. T. and Subrahmanian, V. S. 1992. Probabilistic logic programming. *Information and Computation 101,* 2, 150–201.

Ngo, L. and Haddawy, P. 1997. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science 171,* 1–2, 147–177.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Pinto, J. and Reiter, R. 1993. Temporal reasoning in logic programming: A case for the situation calculus. In *Proc. of the International Conference on Logic Programming*. 203–221.

Poole, D. 1993. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64 64,* 1, 81–129.

Poole, D. 1997. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence 94,* 1-2, 7–56.

Przymusinski, T. C. 1991. Stable semantics for disjunctive programs. *New Generation Computing 3/4*, 401–424.

Riguzzi, F. 2004. Learning logic programs with annotated disjunctions. In *14th Internation Conference on Inductive Logic Programming (ILP2004), Porto, 6-8 September 2004*, A. Srinivasan and R. King, Eds. Springer Verlag, Heidelberg, Germany, 270–287.

Riguzzi, F. 2007. A top down interpreter for LPAD and CP-logic. In *The 14th RCRA workshop Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*.

Sakama, C. and Inoue, K. 1994. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of automated reasoning 13,* 1, 145–172.

Santos Costa, V., Page, D., Qazi, M., and Cussens, J. 2003. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2003)*. Morgan Kaufmann, 517–524.

Sato, T. and Kameya, Y. 1997. PRISM: A language for symbolic-statistical modeling. In *Proceedings of the International Joint Conferences on Artificial Intelligence, IJCAI'97*. 1330–1335.

Shafer, G. 1996. *The art of causal conjecture*. MIT Press.

Tran, N. and Baral, C. 2004. Encoding probabilistic causal model in probabilistic action language. In *AAAI*.

Van Belleghem, K., Denecker, M., and De Schreye, D. 1997. On the Relation between Situation Calculus and Event Calculus. *Journal of Logic Programming, special issue on Reasoning about Actions and Change 31,* 1-3, 3–37.

Van Gelder, A., Ross, K., and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

Venn, J. 1866. *The Logic of Chance: An Essay on the Foundations and Province of the Theory of Probability*.

Vennekens, J., Denecker, M., and Bruynooghe, M. 2006. Representing causal information about a probabilistic process. In *Logics in Artificial Intelligence, 10th European Conference, JELIA'06, Proceedings*. Lecture Notes in Computer Science, vol. 4160. Springer, 452–464.

Vennekens, J., Verbaeten, S., and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *Logic Programming, 20th International Conference, ICLP'04, Proceedings*. Lecture Notes in Computer Science, vol. 3132. Springer, 431–445.