

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

Abstract

We examine the relation between constructive processes and the concept of causality. We observe that causality has an inherent dynamic aspect, i.e., that, in essence, causal information concerns the evolution of a domain over time. Motivated by this observation, we construct a new representation language for causal knowledge, whose semantics is defined explicitly in terms of constructive processes. This is done in a probabilistic context, where the basic steps that make up the process are allowed to have non-deterministic effects. We then show that a theory in this language defines a unique probability distribution over the possible outcomes of such a process. This result offers an appealing explanation for the usefulness of causal information and links our explicitly dynamic approach to more static causal probabilistic modeling languages, such as Bayesian networks. We also show that this language, which we have constructed to be a natural formalization of a certain kind of causal statements, is closely related to logic programming. This result demonstrates that, under an appropriate formal semantics, a rule of a normal, a disjunctive or a certain kind of probabilistic logic program can be interpreted as a description of a causal event.

1 Introduction

The topic of this paper is the concept of *causality* and its relation to constructive processes. To motivate our interest in this, let us consider a causal statement of the following form:

Pneumonia causes chest pain. (1)

As a starting point, we present an analysis of the intuitive meaning of this statement. On the one hand, there is the rather obvious component that patients with pneumonia

*Research supported by GOA 2003/8 Inductive Knowledge Bases and by FWO Vlaanderen.

[†]Joost Vennekens is a postdoctoral researcher of the FWO.

also suffer from chest pain. On the other hand, there are also two more subtle aspects of this statement.

- In essence, the causal statement is *dynamic* in nature, i.e., it refers to something that might happen, to a kind of occurrence, to an activation of some mechanism in the domain of discourse. In the case of statement (1), we are talking about a biological process involving viruses, lung tissue, nerves, and impulses. Once the patient has pneumonia, this mechanism is put into motion and eventually produces the phenomenon of chest pain. Because such a process might take some time to complete, this implies that if we were to observe pneumonia *now*, then the accompanying chest pain might only manifest itself *later*. This is in contrast to, for instance, the statement “pneumonia implies chest pain”, which is static in nature and refers to a single point in time; i.e., if we observe pneumonia now, then the implied chest pain is also observed now. In what follows, we will refer to such mechanisms, to these “things that might happen”, as *events*. We remark that this use of the term “event” is different from that in probability theory, where it typically denotes a possible outcome of an experiment (e.g., a coin landing on heads); in our use of this term, it would rather be the actual execution of the experiment itself (e.g., tossing the coin) that is called an event.
- Saying that pneumonia causes chest pain also implies that chest pain is a property that *needs* to be caused, i.e., the statement suggest that there exists some *a priori* state of affairs and that, in this state, chest pain is absent. It is only if the original state is affected in some way—for instance, if the mechanism by which pneumonia causes chest pain is activated—that the initial state of this property might change and the patient might suffer chest pain.

These observations suggest that if we have an exhaustive enumeration of all causal statements that are relevant with respect to some particular (aspect of a) domain of discourse, we could predict the final states in which this domain might end up by considering which possible sequences of events might occur. For instance, let us look at the following set of causal statements:

Pneumonia causes chest pain. (2)

Chest pain causes insomnia. (3)

Insomnia causes headache. (4)

Initially, the properties that might be caused (chest pain, insomnia, and headache) are all in their original state of being absent. If the patient now has pneumonia, then the event described by (2) will occur and cause chest pain, after which event (3) will also occur, causing insomnia, which finally also causes (4), thus resulting in a patient with a headache. On the other hand, for a patient without pneumonia, none of these events will be caused, so—if the enumeration really is exhaustive—there will be no headache.

Each of the causal statements we have considered so far describes an event whose effect is known with certainty. In real life, few events have this property. For instance, it is easy to image that the biological process underlying statement (1) (i.e., the virus infecting the lung tissue, which causes an nerve impulse to brain, which leads to the

phenomenon of pain) might actually sometimes not result in the patient feeling chest pain at all. To take into account such non-determinism, we say, for instance:

Pneumonia *might* cause chest pain.

For another example, the following statement describes the possible effects of a risky surgical procedure:

The surgery might cause the recovery of the patient, but it also might cause his death.

In both these cases, it is natural to quantify the uncertainty by assigning a probability to the possible outcome(s), such as:

Pneumonia causes chest pain with probability 0.8.

or:

With probability 0.7, the surgery causes the recovery of the patient, but with probability 0.3, it causes his death.

Statements of this form will be our central topic. For obvious reasons, we will refer to such a statement as a *causal probabilistic event description*, or also, somewhat less accurately, simply as a *causal probabilistic event*, which we abbreviate as *CP-event*.

We will then consider a simple knowledge representation language, in which the causal structure of a domain is represented by an enumeration of all relevant CP-events. We call such a set of CP-events a *CP-theory* and refer to the language of all CP-theories as *CP-logic*. As suggested by the above discussion, we will define the semantics of CP-logic by means of certain probabilistic processes, which can be constructed by considering the ways in which the events described by a CP-theory could actually happen. This generalizes in a rather straightforward way the process we described above in the context of the deterministic causal statements (2), (3) and (4); only, instead of each event having just a single effect, which results in a linear progression from an initial state to a unique final state, we now get events that can have many possible effects, which leads to a branching of possibilities, generating a tree-like progression from the initial state to a number of possible final states. We will call the probabilistic processes that can be generated by a given CP-theory its *execution models*.

In general, a CP-theory may have many execution models. Indeed, this is to be expected, since, clearly, certain relevant information about the dynamic evolution of a domain is not expressed in our language; in particular, a CP-theory does not incorporate any temporal information, i.e., it does not specify *when* a particular CP-event might happen, how long such an event will last, or even the order in which events happen. However, as we will show later, all execution models of a CP-theory generate precisely the same probability distribution over their final states. This uniqueness result is an interesting property, because, typically, we are not really interested in the actual details of the evolution of a domain anyway, but only care about the probability of arriving at a certain end result. Our result now shows that causal information in the form of a CP-theory suffices to know which possible outcome will occur with which probability.

This offers an appealing explanation for why causality is such an important concept: causal information is in essence a compact and robust way of specifying just enough properties of the behaviour of a non-deterministic process to uniquely characterize the probability distribution that it generates.

CP-logic is essentially a causal probabilistic modeling language. It mainly distinguishes itself from other such languages by its explicit focus on the dynamic nature of causality—an aspect which is somewhat ignored in current literature. Indeed, let us consider, for instance, Pearl’s influential work on causality (Pearl 2000). In this approach, the causal structure of a domain is described by a causal Bayesian network, i.e., a directed acyclic graph, in which every node corresponds to some random variable. The intuitive reading of such a network is that the value of every node is causally determined by the values of its parents in the graph. We can view such a network as an abstract representation of a class of probabilistic processes, in which, whenever the values of all parents of a node have been determined, an event occurs that propagates these values to the node itself. Here, too, we can make the observation that, in general, such a process is not unique (because for any two nodes with no path between them, the network does not specify which of the events associated to these nodes will happen first), but that all these processes do generate the same distribution.

Now, for many domains, all of the relevant events might not fit directly into the rigid structure that a Bayesian network imposes. This will be the case, for instance, when more than one event is involved in determining the value of a single random variable, or when the propagation of values does not always happen in the same direction. In this sense, CP-logic extends Bayesian networks by allowing a more flexible and fine-grained representation of causal events, in which such phenomena can be modeled in a more direct and straightforward way. The uniqueness result described previously shows that we can do this and still retain the property that every theory generates a single probability distribution over possible final states of the domain.

The version of CP-logic developed in this paper is still restricted in a number of ways. Our primary goal here is to investigate how to construct a knowledge representation language based on our analysis of the kind of causal statements, that we have been discussing in this introduction. As such, our language will be sufficient for this purpose, but not much more than that. In particular, we will keep the representation of the possible effects of events quite rudimentary, assuming, for instance, that different events have neither opposite nor additive effects. Also, we will restrict attention to finite theories only. In this way, we end up with a language that illustrates the points we wish to make about causality, while still being relatively easy to define. We believe that it forms a suitable core language, which could, in future work, easily be extended to cope with a wider range of causal knowledge. In Section 10, this will be discussed further.

In summary, this paper presents the following contributions. We explore the dynamic nature of causality and develop a representation language for causal knowledge, based on the construct of a causal probabilistic event. We prove that the information contained in a set of CP-events suffices to be able to predict the end result of the evolution of a domain. This study of causality extends and complements Pearl’s work on this topic in a number of ways: we identify the concept of a CP-event as a unit of causal information, that is more basic than the parents-child relation underlying causal Bayesian networks. In this way, we get a more flexible and fine-grained representation

of causal events, which allows more straightforward, compact, and elaboration tolerant models of causal knowledge. Moreover, we are able to support and clarify Pearl’s claims on the stability of causal information and its importance for achieving compact representations, by showing that causal information captures precisely those aspects of the behaviour of a probabilistic process that are relevant for its final outcome.

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 suggests a way of defining a more general version of CP-logic. This will be done in Section 5. Section 6 then discussed the resulting definitions in more detail. In Section 7, we investigate the precise relation between CP-logic and Bayesian networks. Proof of the theorems presented in this paper will be given in Section A. Section 9 discusses some related work.

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 \perp and a greatest element \top . An operator $O : L \rightarrow L$ 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 $(y, O(y), O(O(y)), \dots)$. In particular, because \perp 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 \perp .

2.2 Some concepts from logic programming

We assume familiarity with classical logic. A *Herbrand* interpretation for an alphabet 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 by itself and each function symbol f/n by the function mapping a tuple (t_1, \dots, t_n) to $f(t_1, \dots, t_n)$. 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 $\text{lub}_{\leq_p}(S)$. The three-valued truth function φ^ν for sentences φ and partial Herbrand interpretations ν is defined by induction:

- $p^\nu = \nu(p)$, for $p \in HB(\Sigma)$;
- $(\psi \wedge \varphi)^\nu = \text{Min}_{\leq_t}(\psi^\nu, \varphi^\nu)$;
- $(\forall x \varphi(x))^\nu = \text{Min}_{\leq_t}(\{\psi(t)^\nu \mid t \in HU(\Sigma)\})$.
- $(\neg\varphi)^\nu = (\varphi^\nu)^{-1}$ 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, Ross, and Schlipf 1991). We present the formalization of this semantics in (Denecker and Vennekens 2007). Formally, a logic program P is a set of rules of the form $p \leftarrow \varphi$, where p is a ground atom and φ is a first-order sentence. 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:

- $\nu^0 = \perp_{\leq_p}$, the mapping of all atoms to \mathbf{u} ;
- $\nu^\lambda = \text{lub}_{\leq_p}(\{\nu^\beta \mid \beta < \lambda\})$, 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 induction sequence of P converges to the well-founded model of P , as it was defined in (Van Gelder, Ross, and Schlipf 1991).*

In certain logic programming variants, such as abductive logic programs (Kakas, Kowalski, and Toni 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 head of a rule of the program. This distinction is similar to that between endogenous and exogenous predicates, 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 causal probabilistic events

In this section, we formally define the language of CP-logic. We assume that we have a relational vocabulary Σ at our disposal, i.e., a collection of predicate symbols and constants, such that any particular state of our domain of discourse corresponds to a Herbrand interpretation of Σ , i.e., a set of ground atoms. The restriction to Herbrand interpretations is made solely to ease notation; it is easy to extend all our definitions and results to arbitrary domains. We will make the standard distinction between properties that are endogenous (internal) to the causal process being modeled and properties that are exogenous (external); the endogenous properties are those which are affected by the process, while the exogenous properties simply describe the context in which it is taking place. To this end, we split the predicates of our vocabulary into a set of endogenous predicates and a set of exogenous ones.

3.1 Syntax

We want to describe the causes and effects of probabilistic events. A cause for an event will be represented by a first-order sentence φ , i.e., φ does not contain free variables. To represent the effects of an event, we will assume that our vocabulary is constructed in such a way that an event either does not affect the state of the domain at all, or causes a single (property corresponding to a) ground atom with an endogenous predicate to become true. A *CP-event* is then a statement of the following form:

$$(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n) \leftarrow \varphi, \quad (5)$$

where φ is a first-order sentence, the p_i are ground atoms with an endogenous predicate and the α_i are non-zero probabilities with $\sum \alpha_i \leq 1$. Such an expression is read as:

“Property φ causes an event, whose effect is that it makes at most one of the properties p_i becomes true, and for each p_i , the probability of it being caused by this event is α_i .”

If an event has a deterministic effect, i.e., it always causes some atom p with probability 1, we also write simply $p \leftarrow \varphi$ instead of $(p : 1) \leftarrow \varphi$. We allow the precondition φ to be absent, meaning that the CP-event always happens. In this case, the CP-event is called *unconditional* and we omit the ‘ \leftarrow ’-symbol as well.

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

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:

$$(Paresis : 0.25) \leftarrow Syphilis. \quad (6)$$

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.

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.

$$(Angina : 0.2) \leftarrow Pneumonia. \quad (7)$$

$$(Pneumonia : 0.3) \leftarrow Angina. \quad (8)$$

$$(Pneumonia : 0.4) \vee (Angina : 0.1) \leftarrow Infection. \quad (9)$$

Example 3. A CP-theory is a *multiset* of CP-event, 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:

$$(Infection : 0.1). \quad (10)$$

$$(Infection : 0.1). \quad (11)$$

We now define some notation to refer to different components of a CP-theory. The head $head(r)$ of a rule r of form (5) is the set of all pairs (p_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 ground atoms p_i for which there exists an α_i such that $(p_i, \alpha_i) \in head(r)$. Similarly, by $body_{At}(r)$ we will denote the set of all ground atoms p which “appear”¹ in $body(r)$. For the above Example 2, if r is the CP-event (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-event $E \leftarrow \varphi$ a *rule* if we want to emphasize that we are referring to a syntactical construct. We also introduce the concept of a *non-ground* rule as a way of concisely representing sets of CP-events with identical structure. Concretely, such a non-ground rule is of the form:

$$\forall \mathbf{x} (A_1 : \alpha_1) \vee \dots \vee (A_n : \alpha_n) \leftarrow \varphi,$$

where the atoms A_i and the formula φ now may contain free variables, taken from the universally quantified tuple of variables \mathbf{x} . Such a non-ground rule is seen as an abbreviation for the set of all rules $r[\mathbf{x}/\mathbf{t}]$ that result from replacing the variables \mathbf{x} by a tuple \mathbf{t} of ground terms in alphabet 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-events

$$\begin{aligned} (Angina(John) : 0.2) &\leftarrow Pneumonia(John). \\ (Angina(Mary) : 0.2) &\leftarrow Pneumonia(Mary). \end{aligned}$$

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.

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 regular CP-events. When using such non-ground rules in examples, we will implicitly assume that predicate 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.

3.2 Semantics

This section defines the formal semantics of CP-logic. A CP-theory expresses certain knowledge about the dynamic evolution of a domain. To make this more formal,

¹More 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 $p(\mathbf{t})$ a ground atom, $At(p(\mathbf{t})) = \{p(\mathbf{t})\}$;
- For $\varphi \circ \psi$, with \circ either \vee or \wedge , $At(\varphi \circ \psi)$ is defined as $At(\varphi) \cup At(\psi)$;
- For $\neg\varphi$, $At(\neg\varphi) = At(\varphi)$;
- For $\Theta x \varphi$, with Θ either \forall or \exists , $At(\Theta x \varphi) = \cup_{t \in H_U(\Sigma)} At(\varphi[x/t])$, where $H_U(\Sigma)$ denotes the Herbrand universe for the alphabet Σ .

we will assume that this evolution corresponds to a simple kind of probabilistic process, similar to, e.g., the processes considered in (Halpern and Tuttle 1993). Concretely, a process starts in some initial state and, through a sequence of possibly non-deterministic events, it probabilistically progresses towards any of a number of possible final states.

Our basic mathematical object will be that of a tree structure, in which the edges are labeled with probabilities. Each node in this tree corresponds to a state of the domain, with the root representing its initial state and the leaves its possible final states. Formally, we will assume a function \mathcal{I} that maps each node s to a Herbrand interpretation $\mathcal{I}(s)$, which represents the state of the domain to which this node corresponds.

Definition 1. Let Σ be a vocabulary. A *probabilistic Σ -process* \mathcal{T} is a pair $\langle T; \mathcal{I} \rangle$, such that:

- 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 of Σ .

If we interpret the probability associated to an edge (s, s') as the probability of making a transition from s to s' and assume that all these transitions are probabilistically independent, then 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 \perp of the tree, $\mathcal{P}(\perp) = 1$ 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 \perp 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.

Definition 2. 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 T .

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

Definition 3. 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-event of the form:

$$(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n) \leftarrow \varphi.$$

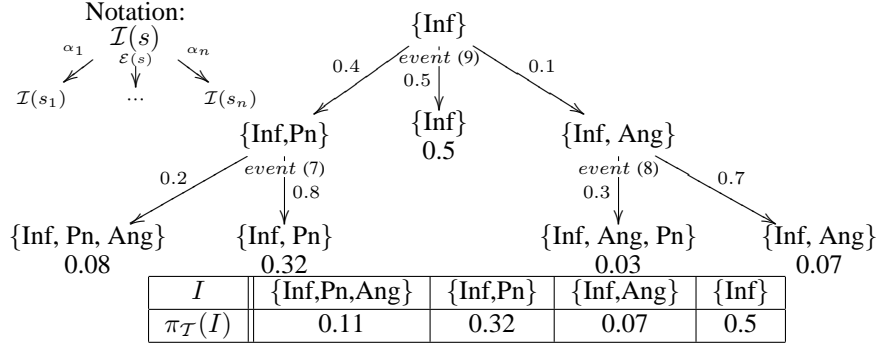


Figure 1: A process \mathcal{T} for Example 2 and its distribution $\pi_{\mathcal{T}}$.

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

- For all $1 \leq i \leq n$, $\mathcal{I}(s_i) = \mathcal{I}(s) \cup \{p_i\}$ 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-event $r \in C$. Because, in our terminology, an event is something that happens at most once, the following definition will also consider, for a node s , the set of all events that have not yet happened 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 4 (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 \perp of \mathcal{T} , $\mathcal{I}(\perp) = X$;
- In each non-leaf node s , an event $\mathcal{E}(s) \in \mathcal{R}_{\mathcal{E}}(s)$ happens, such that its precondition is satisfied in s , i.e., $\mathcal{I}(s) \models \text{body}(\mathcal{E}(s))$;
- For each leaf l of \mathcal{T} , there are no events $r \in \mathcal{R}_{\mathcal{E}}(s)$ for which $\mathcal{I}(l) \models \text{body}(r)$.

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 $X = \{\text{Infected}\}$ is depicted in Figure 1. In general, however, execution models are not unique. Let us illustrate this by another example.

Example 4. John and Mary are each holding a rock. John will throw his rock at a window. With probability 0.5, Mary will also throw her rock at this window. With probability 0.6, John’s rock will hit the window, causing it to break, whereas Mary throwing her rock will cause the window to break with probability 0.8.

$$(Break : 0.8) \leftarrow Throws(Mary). \quad (12)$$

$$(Break : 0.6) \leftarrow Throws(John). \quad (13)$$

$$(Throws(Mary) : 0.5). \quad (14)$$

$$Throws(John). \quad (15)$$

This example has a number of different execution models. Two of these are depicted in Figure 2. We observe that, even though in these two processes events happen in a different order, they produce precisely the same probability distribution. This is a general property of positive CP-theories.

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.1. □

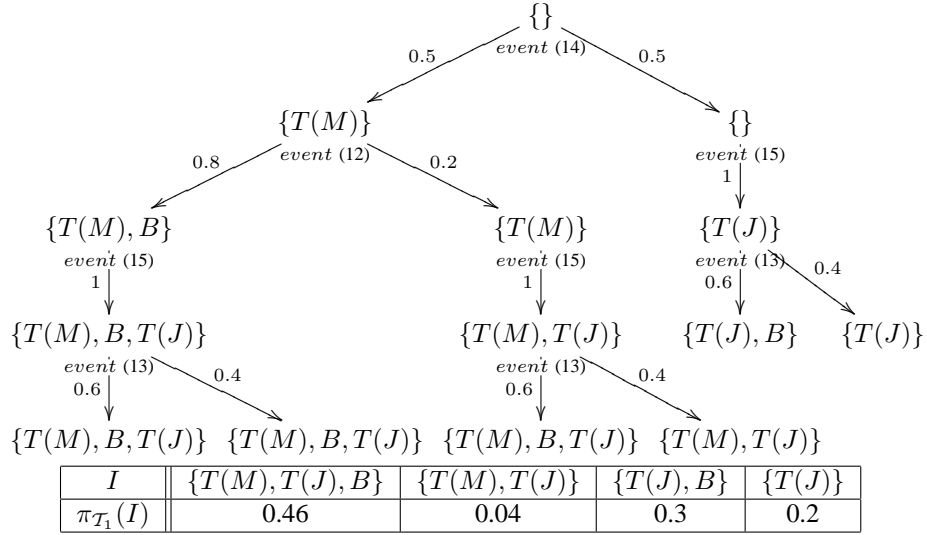
As Example 4 illustrates, the causal information expressed by a CP-theory typically does not suffice to completely characterize a single probabilistic process, i.e., a CP-theory specifies some aspects of the behaviour of such a process, but not all. The above theorem now tells us that, as long as we are only interested in the end result of the process, all aspects that are not specified are actually irrelevant. 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 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.

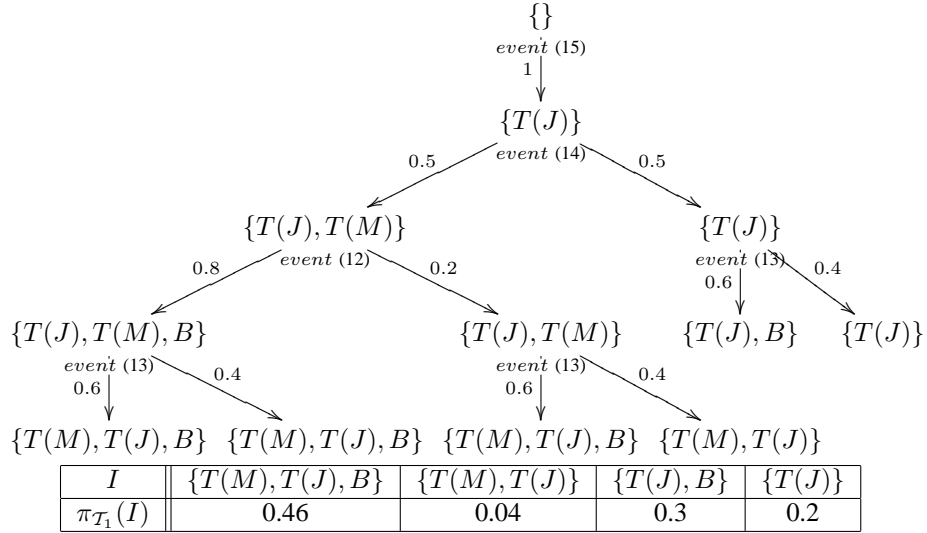
Second, in our construction of CP-logic, we have focused on the dynamic aspect of causality, which has motivated us to define the semantics of this language in terms of probabilistic processes. In this respect, CP-logic differs from the more common approach of, e.g., Bayesian networks, in which causal information is viewed as a description of a probability distribution over possible states of a domain. 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 5. 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.



(a) T_1 : Mary throws—Mary's rock hits—John throws—John's rock hits.



(b) T_2 : John throws—Mary throws—Mary's rock hits—John's rock hits.

Figure 2: Two processes for Example 4.

Definition 6. 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 every execution model of a CP-theory satisfies the following four properties, which seem to be fundamental principles of causal reasoning.

- The principle of *universal causation* states that an endogenous property can only be true if it has been caused by some event, i.e., all changes to the endogenous state of the domain must happen as the consequence of an event.
- The principle of *sufficient causation* states that if an event has a cause, then it must eventually occur.
- The principle of *no deus ex machina events* states that events do not happen spontaneously, i.e., an event can only occur if there is a cause for this and, moreover, events cannot cause themselves. This is a fundamental principle of causal reasoning, that goes back as far as Aristotle.
- The principle of *independent causation* states that every event affects the state of the world in a probabilistically independent way, i.e., knowing the outcome of one event does not give any information about the outcome of a different event. This principle ensures the modularity and robustness of the representation.

We now turn our attention to the question of whether we can extend the above definitions to the case where negation might appear in a CP-theory. However, this extension only makes sense in light of a certain modeling methodology for CP-logic, which we therefore first need to explain.

4 Modeling more complex processes in CP-logic

The exposition of CP-logic in the previous section has been based on rather simple intuitions: we assumed that we were interested in describing a probability distribution over the final states of a causal process, and for every endogenous property of this process, we assumed that it would initially be false and that, if it were ever initiated, it would remain true until the final state. The examples we have encountered so far have all fitted quite naturally into this setup. However, it is easy to imagine that sometimes we might also be interested in intermediate states of a process, or in a process where properties that have been caused can afterwards again be terminated. In this section, we will demonstrate certain modeling methodologies, which allow CP-logic to also be used in these contexts.

Example 5. 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:

$$(LungDamage : 0.1) \leftarrow Pneumonia. \quad (16)$$

$$(Pneumonia : 0.3). \quad (17)$$

The problem with this theory is that, under the natural interpretation of the predicates, it violates the assumptions made by CP-logic: in particular, after pneumonia has been initiated and has caused 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 event (16) 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 our formal semantics, 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.

We remark that, in this example, the correspondence between the states of the formal execution model and the states of the real world is subtly different than it was before: whereas previously, each of our formal states precisely matched to 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 6. A patient is admitted to hospital with pneumonia and stays there for a number of days. At 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. In this case, we are not only interested in the final state of the patient after n days, but also in the evolution of the patient during this period.

On the one hand, this example describes a progression through a sequence of days. On the other hand, for each day, it also describes an event that takes place entirely during this one particular day. In general, a process of this kind will look something like Figure 3: 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 properties at different time points, i.e., we can no longer represent every relevant property

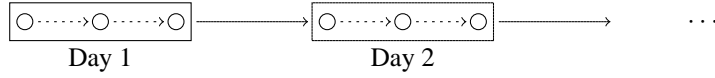


Figure 3: A global process as a sequence of local processes.

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 an alphabet 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 6, we could construct an alphabet 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 alphabet, we can now model Example 6. We assume a fixed range $1..n$ of days, to ensure finiteness of the grounded theory.

$$Pneumonia(1). \quad (18)$$

$$\forall d (Pneumonia(d+1) : 0.8) \leftarrow Pneumonia(d). \quad (19)$$

$$\forall d (Chestpain(d) : 0.6) \leftarrow Pneumonia(d). \quad (20)$$

Here, the CP-events described by (19) are of the kind that propagate from one time point to a later time point, whereas (20) 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 if we measured time in seconds instead of in days, these events would no longer be instantaneous.

According to the informal description of Example 6, the intended model is the process shown in Figure 4. 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 that are instantiations of (19) happen before the instantiations of (20). 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.

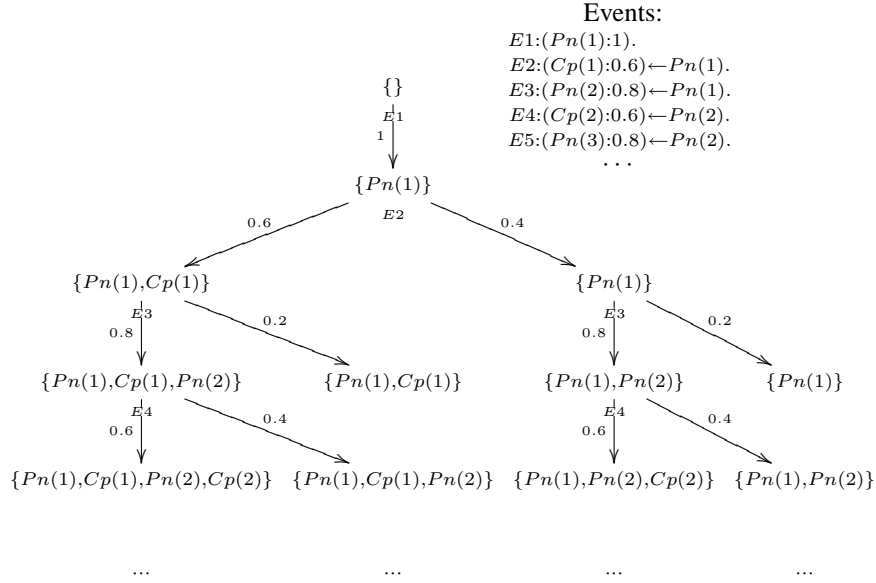


Figure 4: Initial segment of the intended model of Example 6.

Let us now make the above discussion more formal. We assume that, when constructing the alphabet 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 alphabet, 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-events we wrote for this example, we observe that, whenever there is an atom in the head of a CP-event 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-event goes backwards in time. We also remark that if the equality $i = j$ holds, the event is instantaneous (w.r.t. this particular granularity of time), i.e., it is one of those events that takes place entirely within a single time point.

Definition 7. 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-event r happens at time point i , then we would expect i to

lie somewhere between the maximum of all $\lambda(b)$ for which $b \in \text{body}_{At}(r)$, and the minimum of all $\lambda(h)$ for which $h \in \text{head}_{At}(r)$. For a rule r , we write $t_\lambda(r)$ to denote this interval, i.e.,

$$t_\lambda(r) = \left[\max_{b \in \text{body}(r)} \lambda(b), \min_{h \in \text{head}_{At}(r)} \lambda(h) \right].$$

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-event r that actually happens 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 an event 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 event should then happen.

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

Definition 8. Let Σ be an alphabet 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 events of \mathcal{T} happen in the order imposed by κ , i.e., if for all successors s' of a node s , $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$. If this is the case, we also say that \mathcal{T} follows λ .

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 6, the process shown in Figure 4 is an execution model that follows the timing λ specified above.

In the sequel, we will refer to CP-theories, for whose alphabet 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 considered positive formulas as preconditions to events. In this section, we examine whether it is possible to relax this requirement. We first look at a small example.

Example 7. Having pneumonia causes a patient to receive treatment with probability 0.95. Untreated pneumonia causes fever with probability 0.7.

$$(Fever : 0.7) \leftarrow Pneumonia \wedge \neg Treatment. \quad (21)$$

$$(Treatment : 0.95) \leftarrow Pneumonia. \quad (22)$$

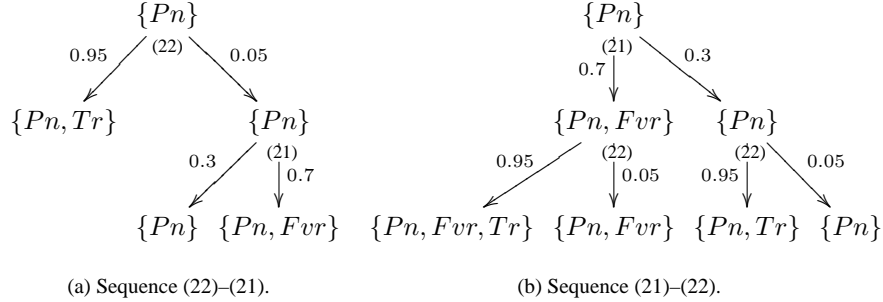


Figure 5: Two processes for Example 7.

Figure 5 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 5(b), event (21) is caused at a moment when its precondition is not yet in its *final* state. In particular, when (21) happens in the initial state, its precondition $\neg Treatment$ holds, but later on, for instance in the leftmost branch, event (22) causes $Treatment$, thereby falsifying this precondition. And so, in the final state of this branch, we see that $Fever$ holds, while the precondition of the event that caused it no longer holds.

In the 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 an event r will not occur until its precondition is in its final state. More precisely, an event cannot occur *until the part of the process that determines whether its precondition holds is fully finished*. For Example 7, it is clear that only the process in Figure 5(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*: in-

herently, there is nothing wrong with the causal process in Figure 5(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 5(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. There, 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 formally in Section 6.3, temporal precedence will do precisely this—at least, if the events containing negation are not instantaneous. Intuitively, this can be explained as follows. For such a non-instantaneous event, 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 an atom from this precondition must happen before this event itself happens. This is now precisely what temporal precedence assumes. The following example is a variant—or rather, a refinement—of Example 7, which illustrates this.

Example 8. 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 alphabet, the CP-theory of Example 7 respects the timing and correctly models the example. Clearly, the intended model of the theory is now that of Figure 5(a): in this model, events occur in the right temporal order. The process of Figure 5(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 events 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 an event should only happen 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 **t** (guaranteed to remain true), **f** (guaranteed to remain false), and **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 events might still happen, 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 **t** is **f** and then gradually build up the set of unknown atoms by applying this principle. Formally, we consider a sequence $(\nu_i)_{0 \leq i \leq n}$ of three-valued interpretations. Initially, ν_0 assigns **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 $p \in head(r)$ with $\nu_i(p) = \mathbf{f}$, it is the case that $\nu_{i+1}(p) = \mathbf{u}$, while for all other atoms p , $\nu_{i+1}(p) = \nu_i(p)$. A sequence that satisfies these properties is called a *hypothetical derivation sequence* in s . 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 $\varphi^{\nu_s} \neq \mathbf{u}$. We now extend the concept of an execution model to arbitrary CP-theories as follows.

Definition 9 (execution model—general case). Let C be a CP-theory in alphabet 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 4 (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 4, but not necessarily the additional condition imposed above, as a *weak execution model*.

In the case of Example 7, this definition indeed gives us the result described above, i.e., the process in Figure 5(a) is an execution model of the example, while the one in Figure 5(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 **f** to *Treatment* and *Fever*;
- ν_1 assigns **u** to *Treatment*;
- ν_2 assigns **u** to *Fever*, because $(\neg Treatment \wedge Pneumonia)^{\nu_1} = \mathbf{u}$;

As such, the only event that can initially happen is the one by which the patient might receive treatment. Afterwards, in every descendant s of \perp , $\nu_s(Treatment)$ will be either **t** or **f**. In the branch where it is **f**, the event by which the patient might die of untreated pneumonia will subsequently happen. Also in the case of Example 9, the execution model yields the intended causal processes.

The temporal precedence assumption imposes a constraint on the order in which events can occur and hence, on the order in which atoms can be caused to become true. In the case of Example 7, 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 9. 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 request that are not accepted by the master.

$$(Master(S1) : 0.5) \vee (Slave(S1) : 0.5). \quad (23)$$

$$Master(S2) \leftarrow Slave(S1). \quad (24)$$

$$Slave(S2) \leftarrow Master(S1). \quad (25)$$

$$\forall x \forall s (Accepts(x, s) : 0.6) \leftarrow Application(s) \wedge Master(x). \quad (26)$$

$$\forall x \forall s \text{ Accepts}(x, s) \leftarrow Application(s) \wedge Slave(x) \wedge \quad (27)$$

$$Master(y) \wedge \neg Accepts(y, s).$$

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 (23), this means that we cannot say upfront 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 events containing negation are not instantaneous—but not for all.

Example 10. We consider a variant of the problem of Example 9 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 9, 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 would try to model this example by the same CP-theory as Example 9, 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 show later in Example 14.

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. 9) simply coincides with the original one (Def. 4), 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 $\mathcal{I}(s) \models \varphi$, then $\nu_s(\varphi) = \mathbf{t}$.*

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. \square

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 8. We now formally 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 7), 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 $body_{At}^+(r)$ ².

Definition 10. A CP-theory C (with negation) *strictly* respects a timing λ if, for all ground atoms h and b :

- If there is CP-event r with $h \in head_{At}(r)$ and $b \in body_{At}^+(r)$, then $\lambda(h) \geq \lambda(b)$;
- If there is CP-event 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-events with negation are not instantaneous events.

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. \square

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.

²Formally, we define, for all sentences φ , the sets $At^+(\varphi)$ and $At^-(\varphi)$ by simultaneous induction as:

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

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

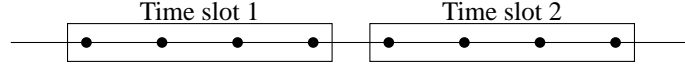


Figure 6: A division into time slots.

Example 11. We consider a time line divided into a number of different time slots, as illustrated in Figure 6. 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.

$$(Send(Client, Req, Server, 1) : 0.7). \quad (28)$$

$$\forall t (Accept(t) : 0.5) \vee (Reject(t) : 0.5) \leftarrow Recvs(Server, Req, t). \quad (29)$$

$$\forall t Send(Server, Answer, Client, t) \leftarrow Accept(t). \quad (30)$$

$$\forall t, s, r, m (Recvs(r, m, t) : 0.8) \vee (Recvs(r, m, t + 1) : 0.1) \leftarrow Send(s, m, r, t). \quad (31)$$

$$\forall t Send(Client, Req, Server, t + 1) \leftarrow Send(Client, Req, Server, t) \wedge \neg Recvs(Client, Answer, t). \quad (32)$$

In this CP-theory, the events described by (28), (29) and (30) all take place inside of a single time slot; the events described by (31) 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 (32) 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 events 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 12. 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.

$$Win(White) \leftarrow \neg Win(Black). \quad (33)$$

$$Win(Black) \leftarrow \neg Win(White). \quad (34)$$

This theory has two weak execution models: one in which event (33) happens first and white wins with probability 1, and one in which (34) happens 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 \perp , $(\neg Win(White))^{\nu\perp} = \mathbf{u} = (\neg Win(Black))^{\nu\perp}$, 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 event is selected before the other. As will be illustrated in Example 14, 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 11. 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 7 and Example 9 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 12. 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 atom 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 11. The theory of Example 9 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 them without explicit time. Below, we illustrate two such cases.

In CP-logic, each atom starts out as false and might become true during 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 13. Consider the following variant of Example 2, in which a doctor can now administer a medicine to suppress chest pain with probability 0.9.

$$Pneumonia(1). \quad (35)$$

$$\forall d (Pneumonia(d+1) : 0.8) \leftarrow Pneumonia(d). \quad (36)$$

$$\forall d (Chestpain(d) : 0.6) \leftarrow Pneumonia(d) \wedge \neg Suppressed(d). \quad (37)$$

$$\forall d Medicine(d) \leftarrow Chestpain(d). \quad (38)$$

$$\forall d (Suppressed(d+1) : 0.9) \leftarrow Medicine(d). \quad (39)$$

$$(40)$$

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 finegrained enough to make all events with negation non-instantaneous³. To illustrate, we consider the following refinement of Example 10.

Example 14. We again consider the setting of Example 10, where the slave does not necessarily wait for the decision of the master, before deciding whether to accept the

³For 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. Truly instantaneous events seem to be limited to, on the one hand, quantum mechanics (if the state of one object collapses, this instantaneously causes the collapse of the state of each entangled object) and, on the other hand, abstract properties defined by social convention (e.g., signing a purchase deed instantaneously makes one the owner of a house).

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 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.

$$\begin{aligned}
& (Master(S1) : 0.5) \vee (Slave(S1) : 0.5) \leftarrow . \\
& Master(S2) \leftarrow Slave(S1). \\
& Slave(S2) \leftarrow Master(S1). \\
& \forall x \forall s \forall t (Decides(x, s, t) : 0.2) \leftarrow Master(x) \wedge Application(s) \wedge \\
& \quad \neg \exists t' (t' < t \wedge Decides(x, s, t')). \\
& \forall x \forall s \forall t (Accepts(x, s, t) : 0.6) \leftarrow Master(x) \wedge Decides(x, s, t). \\
& \forall x \forall s \forall t (Decides(x, s, t) : 0.8) \leftarrow Slave(x) \wedge Application(s) \wedge \\
& \quad \neg \exists t' (t' < t \wedge Decides(x, s, t')). \\
& \forall x \forall s \forall t Accepts(x, s, t) \leftarrow Slave(x) \wedge Decides(x, s, t) \wedge \\
& \quad \neg \exists y \exists t' (Master(y) \wedge t' < t \wedge Accepts(y, s, t')).
\end{aligned}$$

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 (26) and (27) in our original theory from Example 9). 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. Because CP-logic is meant to offer a more fine-grained and flexible representation of causal events than Bayesian networks, we would expect our analysis to show the following. First of all, if all the events in some domain happen to fit directly into the structure imposed by a Bayesian network, then the representation of this domain should be essentially the same in both formalisms. Second, if this is not the case, then we would expect CP-logic to offer some representational advantages.

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 $n_1 = v_1, \dots, n_m = v_m$ 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 $\pi_B(n_1 = v_1, \dots, n_m = v_m)$ of such an assignment of values to all random variables to be rewritten as a product of conditional probabilities $\prod_i \pi_B(n_i = v_i \mid pa(n_i) = \mathbf{v})$,

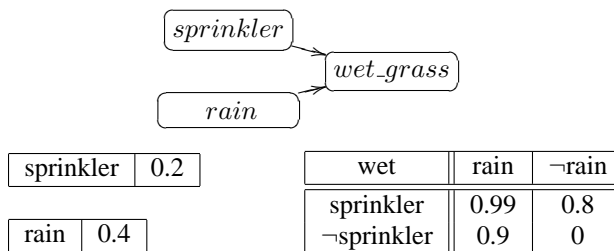


Figure 7: Bayesian network for the sprinkler example.

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 this is done in CP-logic.

7.1 Bayesian networks in CP-logic

As already explained in the introduction, a Bayesian network can also be seen as a description of a class of probabilistic processes. We now 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 13. 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 .

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 15 (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 if both are the case, the probability 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 7. Figure 8 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

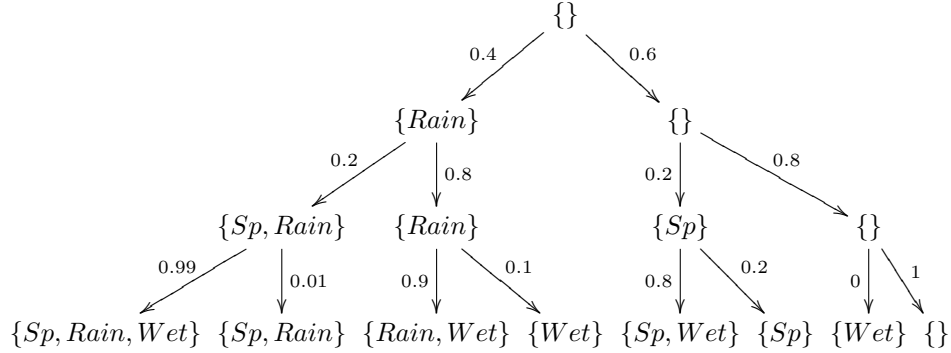


Figure 8: Process corresponding to the sprinkler Bayesian network.

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 14. 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 $\{v_1, \dots, v_k\}$ and parents $p_1 \dots, p_m$ in B , then the children of s in \mathcal{T} are nodes s_1, \dots, 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 $p_1 = w_1, \dots, p_m = w_m$, 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 7 can be transformed into a CP-theory.

Example 15 (Sprinkler—cont'd). We can derive the following CP-theory from the

Bayesian network in Figure 7.

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

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 8 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, \dots, p_m and domain $\{v_1, \dots, v_k\}$, we should construct the set of all rules of the form:

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

where each w_i belongs to the domain of p_i and each α_j is the entry for $n = v_j$, given $p_1 = w_1, \dots, p_m = w_m$ 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 modeling the kind of processes described by a Bayesian network. We will now compare these two formalisms with respect to processes that do not directly fit into the Bayesian network structure. In the introduction, we already mentioned two reasons why this might happen: because multiple events are involved in determining the truth of a single property or because events propagate values in opposite directions.

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 16. 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

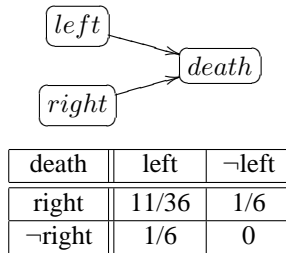


Figure 9: A Bayesian network for Example .

one of its six chambers.

$$(Death : 1/6) \leftarrow Pull_trigger(Left_gun).$$

$$(Death : 1/6) \leftarrow Pull_trigger(Right_gun).$$

Figure 9 shows a Bayesian network for this example. The most obvious difference between these two representations 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-event. 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*⁴ 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.

Finally, we remark that the representation we have discussed in this section is of course only possible if the different causes are truly independent. For instance, in Example 15, *Rain* and *Sprinkler* were not independent as causes for *Wet*, so there we had to stick to a representation that is essentially the same as that of a Bayesian network table.

7.3 Cyclic causal relations

A second reason why a real life process might not correspond directly to a Bayesian network is because it may contain events that propagate values in opposite directions.

⁴The *noisy-or* maps a multiset of probabilities α_i to $1 - \prod_i (1 - \alpha_i)$.

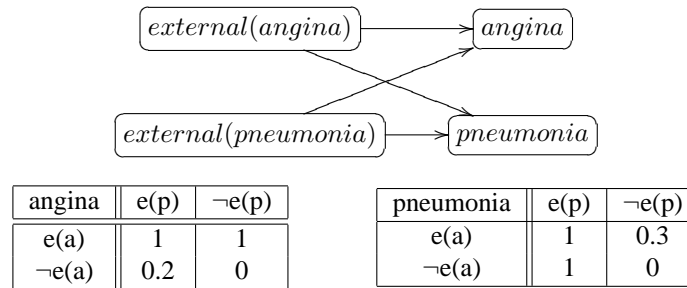


Figure 10: Bayesian network for the *angina-pneumonia* causal loop.

We already saw this in Example 2, where angina could cause pneumonia, but, vice versa, pneumonia could also cause angina. In CP-logic, such causal loops do not require any special treatment. For instance, the loop formed by the two CP-events

$$\begin{aligned}
 (\textit{Angina} : 0.2) &\leftarrow \textit{Pneumonia}. \\
 (\textit{Pneumonia} : 0.3) &\leftarrow \textit{Angina}.
 \end{aligned}$$

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 10. 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.

8 CP-logic and logic programs

In the preceding sections, we identified the concept of a causal probabilistic event as a basic unit of causal information and defined a semantics for sets of such events in

terms of constructive processes. There is an obvious similarity between the syntax of CP-logic and that of logic programs; moreover, our constructive processes 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, Denecker, and De Schreye 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 formalize 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:

$$(h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n) \leftarrow \varphi, \tag{41}$$

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 (41) represents a probability distribution over the following set of logic programming rules:

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

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 15 (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 wanted 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 16 (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 17 (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 now 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 causal probabilistic events.

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

$$\begin{aligned} (P : 0.5) \vee (Q : 0.5) &\leftarrow R. \\ R &\leftarrow \neg P. \\ R &\leftarrow \neg Q. \end{aligned}$$

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 pre-ence 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 9.1. Moreover, they can also be seen as providing a valuable piece of knowledge representation methodology for these languages, as they clarify how causal information can be modeled in these languages. 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:

$$h_1 \vee \dots \vee h_n \leftarrow \varphi. \quad (42)$$

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-event r , the sum of the probabilities α_i appearing in $head(r)$ is precisely 1 and, moreover, every such $\alpha_i > 0$. This is without loss of generality, since every CP-theory can be brought in this form by first adding $(t : 1 - \sum_i \alpha_i)$ to all rules and then dropping all tuples $(h : 0)$. 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 (42) is: “ φ causes a non-deterministic event, whose effect is precisely one of h_1, \dots, h_n .” Clearly, this is a different informal reading than in the standard stable model semantics for disjunctive programs (Przymusiński 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 semantics. Interestingly, the *possible model semantics* (Sakama and Inoue 1994) for disjunctive programs is quite similar to the LPAD treatment. Indeed, 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 $h \leftarrow \varphi$, 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 $h \leftarrow \varphi$ 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 appear in mathematical texts, an inductive definition is represented as a set of definitional rules, which are of the form $\forall \mathbf{x} p(\mathbf{t}) \leftarrow \varphi$, 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 formalize 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 by 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 Causal languages

Our work on CP-logic is primarily intended as a study of the dynamic nature of causal information from a knowledge representation perspective. As such, it is closely related to the work of Pearl. In Section 7, we have already compared CP-logic to Pearl’s formal tool of Bayesian networks and showed that it offers certain representational advantages through its more flexible and fine-grained representation of causal events. We have, however, not yet discussed the fact that the focus of our study is somewhat different from Pearl’s. Indeed, his work focuses on the behaviour of causal models in the presence of *interventions*, i.e., outside manipulations that preempt the normal behaviour of the system. This is a topic that is somewhat orthogonal to our work. Indeed, while Pearl examines interventions in the formal context of structural models (a generalization of Bayesian networks), it seems equally possible to do this in the setting of CP-logic. In fact, this is actually a promising direction for future research. Indeed, in CP-logic, one could consider interventions that preempt, add or replace just a single CP-event. This allows more fine-grained manipulations of a causal model than are possible using Bayesian networks or structural models.

Moreover, one of the interesting uses of interventions is the handling of counterfactuals, which have been used by Halpern to define concepts such as “actual causes” (Halpern and Pearl 2001a) and “explanations” (Halpern and Pearl 2001b). The explicitly dynamic processes of CP-logic 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-event 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.

Another attempt to formalize causal knowledge, which, like CP-logic, also has a close relation to logic programming, are McCain and Turner’s *causal theories* (McCain and Turner 1996). A causal theory is a set of rules $\varphi \leftarrow \psi$, 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 $\varphi \leftarrow \psi$ 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 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 10). 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.

9.2 Probabilistic languages

In this section, we compare CP-logic to a number of probabilistic logic programming languages. Let us first and foremost point out that, in general, our main contribution to this field does not consist of some original new formal language, but, rather, of a new intuitive understanding of the “same old” mathematical objects. Indeed, whereas probabilistic logic programming research typically takes some logic programming variant as its starting point, we have begun from fundamental observations about the nature of causality, without assuming any *a priori* relation to logic programs. Nevertheless, our attempt to formalize intuitions about causal probabilistic events has led to a formal language that is remarkably close to probabilistic logic programming. On the one hand, this demonstrates that probabilistic logic programming can be used to deal with causality. On the other hand, it also allows such languages to be understood, motivated and explained from the ground up, based only on the concept of causal probabilistic events, without any reference to prior logic programming developments.

We will now discuss a number of probabilistic logic programming languages in some more detail.

9.2.1 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 $\sum_{i=1}^n \alpha_i = 1$. 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-event 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-events are conjunctions of literals. Such a CP-event r of the form:

$$(h_1 : \alpha_1) \vee \dots \vee (h_n : \alpha_n) \leftarrow \varphi$$

is then transformed into the set of rules:

$$\left\{ \begin{array}{l} h_1 \leftarrow \varphi \wedge \mathit{Choice}_r(1). \\ \dots \\ h_n \leftarrow \varphi \wedge \mathit{Choice}_r(n). \\ (\mathit{Choice}_r(1) : \alpha_1) \vee \dots \vee (\mathit{Choice}_r(n) : \alpha_n). \end{array} \right\}$$

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 $\mathit{ChoiceRule}(r)$ the last CP-event 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 $\sigma'(\mathit{ChoiceRule}(r)) = (\mathit{Choice}_r(i) : \alpha_i)$. It is quite obvious that this one-to-one correspondence preserves both the probabilities of selections and the (restrictions to the original alphabet 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 alphabet 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 the event-based structure of CP-logic 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.

9.2.2 P-log

P-log (Baral, Gelfond, and Rushton 2004) 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:

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

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 literals⁵. The meaning of a statement of the above form is that, if the body φ of the rule is satisfied, the attribute $A(\mathbf{t})$ will take on a value from the intersection of its domain with the set of all terms x for which $P(x)$ holds. 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:

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

Such a statement should be read as: If φ holds, then the probability of attribute $A(\mathbf{t})$ taking on value y due to the random selection process described by rule r is α .

The information expressed by a random selection rule and its associated probabilistic atoms is somewhat similar to a CP-event, 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, 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:

$$[r] \text{ random}(\text{Leave_through} : \{x : \text{Open_door}(x)\}).$$

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 actions. The difference between observations and actions is the same as highlighted by (Pearl 2000), i.e., observations are supposed to have been generated by the causal processes described by the theory, whereas actions explicitly interfere with the normal state of affairs and,

⁵An 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.

therefore, cannot and should not be explained by the theory. As such, the scope of P-log is significantly broader than that of CP-logic and it is clearly a more full-blown knowledge representation language than CP-logic, which is only aimed at expressing a specific kind of causal knowledge.

9.2.3 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

$$P(\mathbf{t}_0) \mid B_1(\mathbf{t}_1), \dots, B_n(\mathbf{t}_n).$$

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 $P(\mathbf{t}_0) \mid B_1(\mathbf{t}_1), \dots, B_n(\mathbf{t}_n)$ 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, Blockeel, Bruynooghe, and Ramon 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 Relation 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 processes and 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, for instance, 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 16. In a BLP, the relation between the guns firing and the player's death could be represented by the following clause:

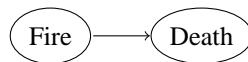
$$Death \mid Fire(x).$$

	$Fire(x) = \mathbf{t}$	$Fire(x) = \mathbf{f}$
$Death = \mathbf{t}$	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:

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



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.

9.2.4 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, Friedman, Koller, and Pfeffer 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) (Costa, Page, Qazi, and Cussens 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, Kimmig, and Toivonen 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:

$$\alpha : h \leftarrow b_1, \dots, b_n,$$

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

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

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:

$$0.8 : \text{likes}(X, Y) \leftarrow \text{likes}(X, Z), \text{likes}(Z, Y),$$

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

We have presented a probabilistic logic study of causality. Our study started from the observation that causality has an inherent dynamic aspect, which leads naturally to the notion of a *causal probabilistic event description* as a fundamental unit of causal knowledge. We have formalized the semantics of such events in terms of constructive probabilistic processes. We then obtained a theorem which, in our opinion, explains why causality is so important in common sense reasoning: the possible outcomes of these causal processes are independent of the order in which events happen. So, apparently, causal statements contain “just enough” information to predict the outcome of such processes.

This theorem holds universally for positive CP-theories, but to obtain it for CP-theories with negation as well, we had to further constrain the class of causal processes we consider by imposing the *temporal precedence assumption*, which states, roughly speaking, that an event can only happen after the part of the process that determines whether its precondition holds is completely finished. As we have shown, this saves our uniqueness theorem, and does this in such a way that temporal theories are treated correctly, i.e., we have proven that if a theory strictly respects some intended timing, then it will have an execution model that follows this timing. This result is particularly useful because it shows that—also in those domains where a naive atemporal encoding would not give the right results—the user can always produce a correct theory by making time explicit. This will also be appropriate if the causal processes are too complex to be modeled without time (cf. Example 10) or if we care about the intermediate states of the process. As we showed, whether to make time explicit is not an all-or-nothing question, since a mixture is also possible, allowing, for instance, to represent a global timed process that consists of a sequence of local “instantaneous” processes.

On a different level, this paper also contributes as a unifying semantic study of existing probabilistic and non-probabilistic logics and formalisms. We showed how

CP-logic refines causal bayesian networks and several logics based on them. We also elaborated 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.

The primary intention of the present study has been to investigate and formalize the dynamic nature of causality. To simplify discussions, we kept the language of CP-logic as simple as possible. As became apparent in the comparison with other logics, e.g., P-log, CP-logic lacks expressivity in several respects to be truly useful for a broad class of applications. To make CP-logic 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 causal probabilistic events. 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 reenforce 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 formalize 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 $\pi_C(\varphi)$ 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.

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 9 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-events $r \in C$ 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 interpretations 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 ν 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 straightforward 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 $\text{body}(r)^{\nu_i} \neq \mathbf{f}$. Since $\nu \leq_p \nu_i$, this would imply that $\text{body}(r)^\nu \neq \mathbf{f}$, 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, $\nu_s \in O_s$. \square

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 . 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. \square

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 $\text{dom}(\sigma) \subseteq C$ to pairs $(p : \alpha) \in \text{head}^*(r)$. The probability function of selections can be extended to partial selections by setting $P(\sigma) = \prod_{r \in \text{dom}(\sigma)} \sigma^\alpha(r)$. 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 $p \in \text{head}^*(r)$ that was selected for s_{i+1} . Moreover, we have

$$\mathcal{P}(s) = P(\sigma(s)) = \sum_{\sigma' \in S(\sigma(s))} P(\sigma'). \quad (43)$$

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:

- $K_0 = \perp$, 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 $\sigma \in S(s)$, $(K_j)_{j=0}^{2n+1}$ is a well-founded induction sequence 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 sequence of any instance C^σ . For $n > 0$, the induction hypothesis states that $(K_j)_{j=0}^{2n-1}$ is a well-founded induction sequence of all instance 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, $\text{body}(r)$ is true in $K_{2n-1} = \nu_{s_{n-1}}$. Clearly, for each $\sigma \in S(\sigma(s))$, C^σ contains the rule $p \leftarrow \text{body}(r)$. 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 , $\text{body}(r) = \varphi$ is false in K_{2n+1} . \square

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 $r \in \mathcal{E}(l)$, $\text{body}(r)$ 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 . \square

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 , we associate to each node s of \mathcal{T} the set $S(\sigma(s))$ of all those C -selections σ that extend $\sigma(s)$. It is easy to see that, with $L_{\mathcal{T}}$ the set of all leaves of \mathcal{T} , the class $\{S(\sigma(l)) \mid l \in L_{\mathcal{T}}\}$ 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 $\mathcal{I}(l) = J$. By Proposition 3, the well-founded model of a selection $\sigma \in S(\sigma(l))$ for a leaf l is $\mathcal{I}(l)$. Therefore, for a given two-valued interpretation J , the class $\{S(\sigma(l)) \mid l \in L_{\mathcal{T}}(J)\}$ is a partition of the collection $S(J)$ of selections σ such that $WFM(C^\sigma) = J$. This now allows us to derive the following equation:

$$\begin{aligned} \mu_C(J) = \sum_{\sigma \in S(J)} P(\sigma) &= \sum_{l \in L_{\mathcal{T}}(J)} \sum_{\sigma \in S(\sigma(l))} P(\sigma) \\ &= \sum_{l \in L_{\mathcal{T}}(J)} \mathcal{P}(l) && \text{(see equation (43))} \\ &= \pi_{\mathcal{T}}(J). \end{aligned}$$

\square

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 repects 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 8 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 follows κ 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 $r \in \mathcal{E}(s)$ and $\mathcal{I}(s) \models body(r)$, i.e., r is an events 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 $r \in \mathcal{E}(l)$ 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 $r \in \mathcal{E}(s)$ 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 $r \in \mathcal{E}(s)$ 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 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} , $\nu_s(\text{body}(\mathcal{E}(s))) = \mathbf{t}$. 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 $r \in \mathcal{E}(s)$ for which $\mathcal{I}(s) \models \text{body}(r)$ 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 $p \in \text{body}_{At}^-(r)$ 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 $p \in \text{body}_{At}^+(r)$, $\mathcal{I}(s)(p) \leq_t \nu_s(p)$. By a well-known monotonicity property of three-valued logic, $\mathbf{t} = \text{body}(r)^{\mathcal{I}(s)} \leq_t \text{body}(r)^{\nu_s}$. Hence, $\text{body}(r)^{\nu_s}$ is indeed \mathbf{t} . \square

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*, pp. 219–226.
- Baral, C., M. Gelfond, and N. Rushton (2004). Probabilistic reasoning with answer sets. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*, Volume 2923 of *Lecture notes in artificial intelligence (LNAI)*, pp. 21–33. Springer-Verlag.
- Blockeel, H. and W. Meert (2007). Towards learning non-recursive LPADs by transforming them into Bayesian networks. In *Inductive Logic Programming, ILP'06, Revised Selected Papers*, Volume 4455 of *Lecture Notes in Computer Science*, pp. 94–108.
- Breese, J. (1992). Construction of belief and decision networks. *Computational intelligence* 8(4), 624–647.
- Costa, V. S., D. Page, M. Qazi, and J. Cussens (2003). CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, pp. 517–524. Morgan Kaufmann.
- Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 115–122. Morgan Kaufmann.

- De Raedt, L., A. Kimmig, and H. Toivonen (2007). ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pp. 2462–2467.
- Dekhtyar, A. and V. Subrahmanian (2000). Hybrid probabilistic programs. *Journal of Logic Programming* 43(3), 187–250.
- Denecker, M. and E. Ternovska (2007). A logic of non-monotone inductive definitions. *Transactions On Computational Logic (TOCL)*.
- Denecker, M. and J. Vennekens (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*, Volume 4483 of *Lecture Notes in Artificial Intelligence*, pp. 84–96. Springer.
- Fierens, D., H. Blockeel, M. Bruynooghe, and J. Ramon (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*, Volume 3625 of *Lecture Notes in Computer Science*, pp. 121–135. Springer.
- Finzi, A. and T. Lukasiewicz (2003). Structure-based causes and explanations in the independent choice logic. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence, UAI'03*.
- Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (2001). Learning probabilistic relational models. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, pp. 7–34. Springer-Verlag.
- Ghahramani, Z. (1998). Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures*, Lecture Notes in Artificial Intelligence, pp. 168–197. Springer-Verlag.
- Halpern, J. and J. Pearl (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 J. Pearl (2001b). Causes and explanations: A structural model approach – part II: Explanations. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, UAI'01*.
- Halpern, J. and M. Tuttle (1993). Knowledge, probability, and adversaries. *Journal of the ACM* 40, 917–960.
- Jaeger, M. (1997). Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*.
- Kakas, A. C., R. Kowalski, and F. Toni (1992). Abductive logic programming. *Journal of Logic and Computation* 2(6), 719–770.
- Kersting, K. and L. De Raedt (2000). Bayesian logic programs. In J. Cussens and A. Frisch (Eds.), *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pp. 138–155.
- Kersting, K. and L. De Raedt (2001). Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Germany.

- Kersting, K. and L. De Raedt (2007). Bayesian logic programming: Theory and tool. In L. Getoor and B. Taskar (Eds.), *An Introduction to Statistical Relational Learning*. MIT Press. To appear.
- Lakshmanan, L. and F. Sadri (1994). Probabilistic deductive databases. In M. Bruynooghe (Ed.), *Proceedings of the International Symposium on Logic Programming, ILPS'94*, pp. 254–268. MIT Press.
- 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)*, Volume 2173 of *Lecture Notes in Artificial Intelligence*, pp. 336–350. Springer-Verlag.
- McCain, N. (1997). *Causality in Commonsense Reasoning about Actions*. Ph. D. thesis, University of Texas at Austin.
- McCain, N. and H. Turner (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)*, pp. 460–465. AAAI Press.
- Muggleton, S. (2000). Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence* 5(041), 141–153.
- Ng, R. and V. Subrahmanian (1992). Probabilistic logic programming. *Information and Computation* 101(2), 150–201.
- Ngo, L. and P. Haddawy (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science* 171(1–2), 147–177.
- Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.
- Pinto, J. and R. Reiter (1993). Temporal reasoning in logic programming: A case for the situation calculus. In *Proc. of the International Conference on Logic Programming*, pp. 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.
- Przymusiński, T. C. (1991). Stable semantics for disjunctive programs. *New Generation Computing* 3/4, 401–424.
- Riguzzi, F. (2004, September). Learning logic programs with annotated disjunctions. In A. Srinivasan and R. King (Eds.), *14th International Conference on Inductive Logic Programming (ILP2004)*, Porto, 6-8 September 2004, Heidelberg, Germany, pp. 270–287. Springer Verlag.
- 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 K. Inoue (1994). An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of automated reasoning* 13(1), 145–172.
- Sato, T. and Y. Kameya (1997). PRISM: A language for symbolic-statistical modeling. In *Proceedings of the International Joint Conferences on Artificial Intelligence, IJCAI'97*, pp. 1330–1335.
- Van Belleghem, K., M. Denecker, and D. De Schreye (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., K. Ross, and J. Schlipf (1991). The well-founded semantics for general logic programs. *Journal of the ACM* 38(3), 620–650.