Negation in the Head of CP-logic Rules

Joost Vennekens

joost.vennekens@cs.kuleuven.be
Dept. Computerscience — Campus De Nayer
KU Leuven

Abstract. CP-logic is a probabilistic extension of the logic FO(ID). Unlike ASP, both of these logics adhere to a Tarskian informal semantics, in which interpretations represent objective states-of-affairs. In other words, these logics lack the epistemic component of ASP, in which interpretations represent the beliefs or knowledge of a rational agent. Consequently, neither CP-logic nor FO(ID) have the need for two kinds of negations: there is only one negation, and its meaning is that of objective falsehood. Nevertheless, the formal semantics of this objective negation is mathematically more similar to ASP's negation-as-failure than to its classical negation. The reason is that both CP-logic and FO(ID) have a constructive semantics in which all atoms start out as false, and may only become true as the result of a rule application. This paper investigates the possibility of adding the well-known ASP feature of allowing negation in the head of rules to CP-logic. Because CP-logic only has one kind of negation, it is of necessity this "negation-as-failure like" negation that will be allowed in the head. We investigate the intuitive meaning of such a construct and the benefits that arise from it.

1 Introduction

This paper is part of a long-term research project that aims to develop a *Tarskian view* on Answer Set Programming (ASP). Historically, the origins of ASP lie in the seminal papers by Gelfond and Lifschitz on the stable semantics for normal (1988) and extended logic programs (1991). These papers develop an *epistemic view* on logic programs, in which an answer set is seen as an exhaustive enumeration of a rational agent's atomic beliefs. In this view, an atom A belonging to an answer set X means that the agent believes A; $A \notin X$ means that A is not believed; and $\neg A \in X$ means that a is believed to be false. A rule such as:

$$A \leftarrow B_1, \ldots, B_n, \texttt{not } C_1, \ldots, \texttt{not } C_m. \tag{1}$$

tells the agent that if he believes all of the B_i and does not believe any of the C_j, he should believe A. In addition, the agent also obeys the *rationality principle*, believing only what he has reason to believe. The stable model semantics then computes what a perfectly rational agent would believe under all these rules.

While these epistemic intuitions have played a crucial role in the history of ASP, current practice seems to have largely drifted away from them. In particular,

programs written according to the currently prevalent *Generate-Define-Test*-methodology (GDT) (term coined by Lifschitz, 2002) are typically no longer explicitly concerned with the beliefs of an agent. A typical example is the *graph colouring* problem, in which we *generate* the search space of all assignments of colours to nodes, we *define* that two nodes are in conflict if they share an edge and have the same colour, and then *test* that there are no conflicts. Unlike early ASP examples—such as, e.g., Gelfond's example (1991) of interviewing all students for which we do not *know* whether they are eligible for a grant—the statement of the graph colouring problems is not concerned with anyone's knowledge or beliefs, but only with the objective colour of the nodes.

Suppose now that we have an ASP representation of a purely objective GDT problem, such as graph coloring. How should we intuitively interpret this program? Falling back on the papers by Gelfond and Lifschitz, every single statement in the program will be interpreted as an epistemic statement about some agent's knowledge. Obviously, this is a poor match with the objective intuitions behind the problem. Therefore, an alternative informal semantics is needed, which omits this agent, and explains how rules of the program can be interpreted as statements about the real world, in this same way as formulas in classical first-order logic (FO) are. There are now two important and related questions:

- If we view a semantical object such as an answer set as a representation of an objective state of the world, instead of some agent's beliefs, how should we then interpret a rule such as (1)?
- How does this objective interpretation of ASP compare to the classical way of representing such objective information about the world, namely FO?

An extensive study of these two questions has been performed by Denecker and several coauthors. Recent summaries of these results were published by Denecker et al. (2010) and Denecker et al. (2012). A goal of this research program is to reconstruct ASP as a series of conservative extensions of FO. One of its main achievements has been the development of the language of FO(ID) (Denecker and Ternovska, 2007), which extends FO with a construct for representing *inductive definitions*. FO(ID) can be seen as a variant of ASP, which adheres to a strict objective interpretation of its semantical constructs, i.e., a model of an FO(ID) theory does not represent beliefs, but an objective state of the world.

The language of FO(ID) has been further extended in many ways. This paper is concerned with one particular such extension, namely, *CP-logic* (Vennekens et al., 2009), which extends the inductive definition construct of FO(ID) with a means for expressing non-deterministic choice. One application is to represent non-deterministic inductive definitions. For instance, an execution trace of a non-deterministic Turing machine may be defined by means of a rule that states that if the machine reads a character c in a state s at time α, it will be in a state s' at time $\alpha + 1$, where s' is *one of* the states that it may transition to from (s, c). CP-logic represents such non-determinism by allowing disjunction in the head of rules. This is similar in syntax to the kind of rules allowed by, for instance, the DLV language. This is, therefore, another way in which one of ASP's features can be conservatively added to the classical framework. However,

to correctly formalise non-deterministic inductive definitions, not the minimal model semantics must be used, but the *possible world semantics* of Sakama and Inoue (1994).

A more important application of CP-logic, however, is to represent *probabilistic causal laws*. Such relations have received a great deal of attention in the AI community, especially since the influential work by Pearl (2000) on this topic. As shown by Vennekens et al. (2010), CP-logic can actually be seen as a refinement of Pearl's theory, which allows for a more compact and modular representation of certain phenomena. As an example, consider three gear wheels, each of which has an attached crank that can be used to turn it. The first gear wheel is connected to the second, which is in turn connected to the third, so that in 90% of the cases, when one turns the other also turns; however, there is some damage to the gear wheels' teeth, which in 10% of the cases prevents this. In CP-logic, we can represent this by means of seven independent probabilistic causal laws:

$$Turns(Gear1) \leftarrow Crank_1. \tag{2}$$
$$Turns(Gear2) \leftarrow Crank_2. \tag{3}$$
$$Turns(Gear3) \leftarrow Crank_3. \tag{4}$$
$$(Turns(Gear1) : 0.9) \leftarrow Turns(Gear2). \tag{5}$$
$$(Turns(Gear2) : 0.9) \leftarrow Turns(Gear1). \tag{6}$$
$$(Turns(Gear2) : 0.9) \leftarrow Turns(Gear3). \tag{7}$$
$$(Turns(Gear3) : 0.9) \leftarrow Turns(Gear2). \tag{8}$$

By contrast, Pearl would represent it in a less modular way, by means of three structural equations, each of which defines precisely when a particular gear wheel will turn :

$$Turns(Gear1) := Crank_1 \vee (Crank_2 \wedge Trans_{1,2}) \vee (Crank_3 \wedge Trans_{3,2} \wedge Trans_{2,1})$$
$$Turns(Gear2) := Crank_2 \vee (Crank_1 \wedge Trans_{1,2}) \vee (Crank_3 \wedge Trans_{3,2})$$
$$Turns(Gear3) := Crank_3 \vee (Crank_2 \wedge Trans_{2,3}) \vee (Crank_1 \wedge Trans_{1,2} \wedge Trans_{2,3})$$

CP-logic has certain similarities to P-log, a probabilistic extension of ASP (Baral et al., 2008). However, it differs by its focus on representing individual probabilistic causal laws, as discussed by Vennekens et al. (2010, 2009).

As this example illustrates, a causal law in CP-logic may cause some atom(s) to become true, and it may also fail to do so. What is currently not possible, however, is that such a laws causes an atom to be false. For instance, suppose that the first gear wheel may be locked, in order to prevent it from turning. The current way to represent this would be to replace rules (2) and (5) by:

$$(Turns(Gear1) : 0.9) \leftarrow Crank_1 \wedge \neg Locked(1).$$
$$(Turns(Gear1) : 0.9) \leftarrow Turns(Gear2) \wedge \neg Locked(1).$$

However, this goes against our desire for a modular representation of the individual causal laws. Our goal in the current paper is to extend CP-logic to allow instead

to keep rules (2) and (5) as they are, and instead add a rule:

$$\neg Turns(Gear1) \leftarrow Locked(1).$$

In other words, we will examine how CP-logic can be extended with the familiar ASP feature of *negation in the head* Gelfond and Lifschitz (1991). Again, the traditional ASP interpretation of a classical negation literal is rooted in the epistemic tradition: whereas `not` A means that A is not believed to be true, a classical negation literal $\neg A$ means that A is believed to be false. Since FO(ID) and CP-logic have no beliefs, the only thing that negation *can* mean in this context is that A is objectively false. Nevertheless, as this paper will show, there is still a place for negation-in-the-head in such a logic. Our two main contributions are therefore as follows:

- By adding this additional feature to CP-logic, we extend its ability to represent causal laws in a modular way, as illustrated by the above example.
- From the point of view of the larger research project, negation-in-the-head is an ASP feature that, until now, could not yet be given a place within the FO(ID)/CP-logic framework and its Tarskian semantics. This paper offers one way in which this gap can be filled.

This paper is structured as follows. First, Section 2 recalls the definition of CP-logic. Section 2.1 elaborates further on the role of negation in the current version of CP-logic, before Section 3 then discusses our proposed extension with negation in the head. Several uses of this new feature are then discussed in Sections 4 to 6. Finally, Section 7 discusses the implementation of this new language feature.

2 Preliminaries: CP-logic

A theory in CP-logic consists of a set of rules. These rules are called *causal probabilistic laws*, or *CP-laws* for short, and they are statements of the form:

$$\forall \boldsymbol{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \phi. \tag{9}$$

Here, ϕ is a first-order formula and the A_i are atoms, such that the tuple of variables $\boldsymbol{x}$ contains all free variables in ϕ and the A_i. The α_i are non-zero probabilities with $\sum \alpha_i \leq 1$. Such a CP-law expresses that ϕ causes some (implicit) non-deterministic event, of which each A_i is a possible outcome with probability α_i. If $\sum_i \alpha_i = 1$, then at least one of the possible effects A_i must result if the event caused by ϕ happens; otherwise, it is also possible that the event happens without any (visible) effect on the state of the world. For mathematical uniformity, we introduce the notation $r^{=}$ to refer to r itself if the equality holds, and otherwise to the CP-law:

$$\forall \boldsymbol{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \vee (\text{—} : 1 - \sum_i \alpha_i) \leftarrow \phi.$$

Here, the dash is a new symbol that explicitly represents the possibility that none of the effects A_i are caused. Whenever we add this dash to some set X, it does not change X, i.e., $X \cup \{—\} = X$.

The semantics of a theory in CP-logic is defined in terms of its grounding, so from now on we will restrict attention to ground theories, i.e., we assume that for each CP-law, the tuple of variables $\boldsymbol{x}$ is empty. For now, we also assume that the rule bodies ϕ do not contain negation.

For a CP-law r, we refer to ϕ as the *body* of r, and to the sequence $(A_i, \alpha_i)_{i=1}^n$ as the *head* of r. We denote these objects as $body(r)$ and $head(r)$, respectively.

In CP-laws of form (9), the precondition ϕ may be omitted for events that are vacuously caused. If a CP-law has a deterministic effect, i.e., it is of the form $(A : 1) \leftarrow \phi$, then we also write it simply as $A \leftarrow \phi$.

Example 1. Suzy and Billy might each decide to throw a rock at a bottle. If Suzy does so, her rock breaks the bottle with probability 0.8. Billy's aim is slightly worse and his rock only hits with probability 0.6. Assuming that Suzy decides to throw with probability 0.5 and that Billy always throws, this domain corresponds to the following set of causal laws:

$$(Throws(Suzy) : 0.5). \quad (10) \qquad (Broken : 0.8) \leftarrow Throws(Suzy). \quad (12)$$

$$Throws(Billy). \quad (11) \qquad (Broken : 0.6) \leftarrow Throws(Billy). \quad (13)$$

In causal modeling, a distinction is commonly made between endogenous properties, whose values are completely determined by the causal mechanisms described by the model, and exogenous properties, whose values are somehow determined outside the scope of the model. Following this convention, the predicates of a CP-theory are also divided into exogenous and endogenous predicates. We define the semantics of a theory in the presence of a given, fixed interpretation X for the exogenous predicates.

A second common assumption (see e.g. Hall, 2007) is that each of the endogenous properties has some default value, which represents its "natural state". In other words, the *default* value of an endogenous property is the value that it has whenever there are no causal mechanisms acting upon it. The effect of the causal mechanisms in the model is then of course precisely to flip the value of some of the properties from its default to a *deviant* value.

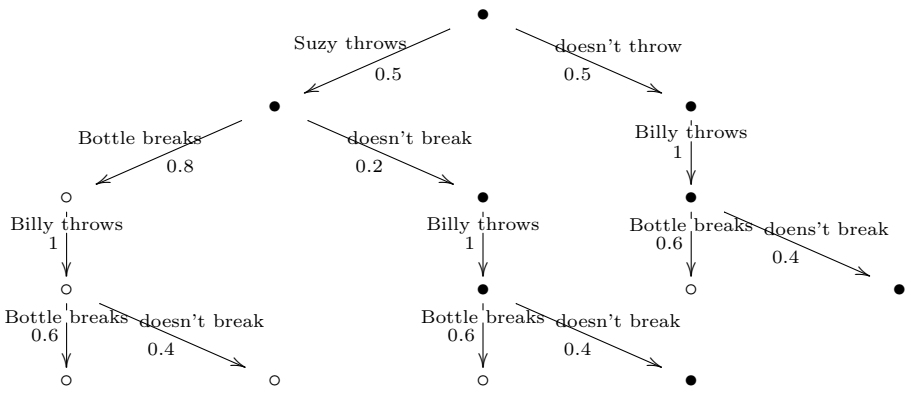
Theories in CP-logic have a straightforward execution semantics. We consider probability trees, in which each node is labeled with an Herbrand interpretation for the endogenous predicates. The root of the tree—i.e., the initial state of our causal process—is labeled with the universally false interpretation $\{\}$. This incorporates our second assumption: w.l.o.g. we force the user to choose his vocabulary in such a way that the default value for each endogenous atom is false. We then constructively extend the tree by applying the following operation as long as possible:

1. Choose a pair (s, r) of a leaf s of the tree and a rule r of the theory, such that $(X \cup \mathcal{I}(s)) \models body(r)$ and there exists no ancestor s' of s such that (s', r) has already been chosen

2. Extend s with children $s_0, \ldots, s_m$, where each s_i corresponds to one of the disjuncts $(h_i : \alpha_i)$ in $head(r^=)$, in the sense that $\mathcal{I}(s_i) = \mathcal{I}_s \cup \{h_i\}$ and the edge from s to s_i is labeled by α.

We call a tree $\mathcal{T}$ constructed in this way an *execution model* of the CP-theory under X. We define a probability distribution $\pi_\mathcal{T}$ over the set of all Herbrand interpretations as: $\pi_\mathcal{T}(I) = \sum_{\mathcal{I}(l)=I} \pi_\mathcal{T}(l)$, where the sum is taken over all leaves l of $\mathcal{T}$ whose interpretation equals I and the probability $\pi_\mathcal{T}(l)$ of such a leaf consists of the product of all probability labels that are encountered on the path to this leaf.

The following picture represents an execution model for the CP-theory of Example 1. The states s in which the bottle is broken (i.e., for which $Broken \in \mathcal{I}(s)$) are represented by an empty circle, and those in which it is still whole by a full one. This pictures does not show the interpretations $\mathcal{I}(s)$; instead, we have just written the effects of each event in natural language as labels on the edges.



The third branch of this execution model consists of five nodes $(s_0, \ldots, s_4)$. The progression of the associated states of the world $(\mathcal{I}(s_0), \ldots, \mathcal{I}(s_4))$ is as follows:

$$(\{\}, \{Throws(Suzy)\}, \{Throws(Suzy)\},$$
$$\{Throws(Suzy), Throws(Billy)\},$$
$$\{Throws(Suzy), Throws(Billy), Broken\}).$$

Note that, in keeping with the Tarskian setting of CP-logic, each interpretation represents an objective state of the world.

Even when starting from the same interpretation X for the exogenous predicates, the same CP-theory may have many execution models, which differ in their selection of a rule to apply in each node (step 1). It was shown by Vennekens et al. (2009) that, because each applicable rule must eventually be applied, the differences between these execution models are irrelevant, as long as we only care about the final states that may be reached. In other words, all execution models $\mathcal{T}$ of the same CP-theory T that start from the same interpretation X generate the same distribution $\pi_\mathcal{T}$. We also denote this unique distribution as π_T^X.

An interesting special case is that in which each rule r is *deterministic*, i.e., it causes a single atom with probability 1. In this case, each execution model is a degenerate tree consisting of a single branch, in which all edges are labeled with probability 1. The successive interpretations in this branch are constructed by adding to the previous interpretation the head of a rule whose body is satisfied. The single leaf of this tree is therefore precisely the least Herbrand model of the set of rules. In this way, positive logic programs and monotone inductive definitions in FO(ID) are embedded in CP-logic.

2.1 Negation in CP-logic

Consider again the role that the CP-law

$$(Broken : 0.9) \leftarrow Throws(Suzy)$$

plays in the above execution model. Initially, when the atom $Throws(Suzy)$ is still at its default, this law is dormant. Once $Throws(Suzy)$ has been caused, this law becomes active and will (eventually) be executed, causing $Broken$ with probability 0.9. Now, suppose we had instead assumed that the default is for Suzy to throw unless she decides to refuse:

$$(Broken : 0.9) \leftarrow \neg RefusesThrow(Suzy).$$
$$(RefusesThrow(Suzy) : 0.5).$$

Under the semantics given so far, this first CP-law would be active in *any* state where $RefusesThrow(Suzy)$ has not deviated from its default. For instance, this law would always be active in the initial state. This means that there would be an execution model in which this law first causes the bottle to break and then, afterwards, Suzy decides to refuse the throw. Such execution models are not very meaningful, or useful.

For this reason, when allowing negation, an additional condition is imposed on the execution models of a CP-theory. The basic idea is to read $\neg A$ not simply as "A is currently at its default vaue", but instead as "A will not deviate from its default". Under this interpretation, the law will only become active once our causal process is far enough along to be able to say with certainty that no deviation will occur. For the above example, this would mean that the first CP-law can only become active *after* the second one has taken place and has failed to cause $RefusesThrow(Suzy)$.

This idea is formalized by means of concepts from three-valued logic, where atoms can be unknown (**u**) in addition to true (**t**) or false (**f**). Given a three-valued interpretation ν, that assigns one of these three truth values to each atom, the standard Kleene truth tables can be used to assign a corresponding truth value $\nu(\phi)$ to each formula ϕ. A two-valued interpretation I is said to be approximated by a three-valued interpretation ν if it can be constructed from it by switching atoms from **u** to **t** or **f**. If I is approximated by ν, then for each formula ϕ, the truth value $\nu(\phi)$ also approximates the truth of ϕ according to I; that is, if $\nu(\phi) = \mathbf{t}$ then $I \models \phi$ and if $\nu(\phi) = \mathbf{f}$ then $I \not\models \phi$.

Now, for each state s of an execution model, we construct an overestimate of the set of atoms that might still be caused in the part of the tree following s. First, the set of events that could potentially happen in this state itself is $Pot(s) = \{r \in \mathcal{R}(s) \mid \mathcal{I}(s) \models body(r)\}$, where $\mathcal{R}(s)$ denotes the set of all rules that have not yet happened in the ancestors of s. For each child s' of s, $\mathcal{I}(s')$ will therefore differ from $\mathcal{I}(s)$ by including at most one atom $A \notin \mathcal{I}(s)$ from the head of one of the rules $r \in Pot(s)$. Therefore, if we construct a three-valued interpretation ν_0 that labels all such atoms A as $\mathbf{u}$ and coincides with $\mathcal{I}(s)$ on all other atoms, then we end up with an approximation of each $\mathcal{I}(s')$ for which s' is a child of s. Now, if an event r is to happen in one of these children s' of s, then it must be the case that that $\mathcal{I}(s') \models body(r)$, which implies that $\nu_1(body(r)) \neq \mathbf{f}$. We now derive a ν_2 from ν_1 by turning into $\mathbf{u}$ all atoms A for which $\nu_1(A) = \mathbf{f}$ and A appears in the head of an r for which $\nu_0(body(r)) \neq \mathbf{f}$. This ν_2 is then an approximation of all $\mathcal{I}(s'')$ for which s'' is a grandchild of s. We can now iterate this principle and construct a sequence $(\nu_1, \nu_2, \ldots)$ of three-valued interpretations, where each ν_i approximates all the $\mathcal{I}(t)$ for which t is a descendant of s, separated from s by at most $i-1$ intermediary nodes. This process will make more and more atoms $\mathbf{u}$, until eventually it reaches a fixpoint, which we denote as $\mathcal{U}(s)$. This fixpoint approximates all the $\mathcal{I}(t)$ for which t is a descendant of s. Therefore, if an atom is $\mathbf{f}$ in $\mathcal{U}(s)$, then it will not be caused anywhere below s.

To illustrate, consider the rightmost branch $(s'_0, s'_1, \ldots, s'_3)$ of the execution model shown in Section 2. The associated three-valued interpretations are as follows, where we abbreviate $Throws$ and $Broken$ by T and B, and $Billy$ and $Suzy$ by By and Sy.

Node s	$\mathcal{U}(s)$		
	t	**u**	**f**
s'_0	$\{\}$	$\{T(Sy), T(By), B\}$	$\{\}$
s'_1	$\{\}$	$\{T(By), B\}$	$\{T(Sy)\}$
s'_2	$\{T(By)\}$	$\{B\}$	$\{T(Sy)\}$
s'_3	$\{T(By)\}$	$\{\}$	$\{T(Sy), B\}$

The following additional condition is now imposed on the execution models of a CP-theory:

> *For a rule r to be allowed to happen in a node s, it is not enough that simply $\mathcal{I}(s) \models body(r)$; in addition, it must also be the case that the truth value of $body(r)$ according to $\mathcal{U}(s)$ is* **t** *instead of* **u**.

Therefore, if the CP-theory of the above example contained an additional rule with body $\neg Throws(Suzy)$, this could be applied from state s'_1 onwards in the above branch, whereas a rule with body $\neg Broken$ would have to wait until s'_3.

With this additional condition, it now becomes possible for execution models to become stuck, in that sense that, in some leaf l, there remain some rules r such that $\mathcal{I}(l) \models body(r)$, yet r cannot happen because $body(r)$ is $\mathbf{u}$ in $\mathcal{U}(s)$. This can happen only when the CP-theory contains loops over negation. Such theories

are viewed as unsound, and no semantics is defined for them. An important class of sound theories are those which are stratified, but there also exist useful sound theories outside of this class (see Vennekens et al. (2009) for a discussion).

Again, an interesting special case is when all rules of the CP-theory are deterministic. In this case, the CP-theory syntactically coincides with a normal logic program, and all of its execution models end in a single leaf l, such that $\mathcal{U}(l)$ is the well-founded model of this program. If the CP-theory is sound, $\mathcal{U}(l) = \mathcal{I}(l)$ is the two-valued well-founded model and therefore also the unique stable model of the program. In this way, normal logic programs with a two-valued well-founded model are embedded in CP-logic. While the limitation to two-valued well-founded models may seem restrictive, in practice this is often mitigated by the fact predicates may be declared as exogenous, which has the same effect as "opening them up" with a loop over negation. Also in FO(ID), definitions whose well-founded model is not two-valued are considered inconsistent, so CP-logic is indeed a true generalization of FO(ID)'s inductive definition construct.

3 Negation in the head

A CP-theory represents a set of causal mechanisms, that are activated one after the other, and together construct the final state of the domain. Each such causal mechanism has the same kind of effect: for some set of atoms, it causes at most one of these atoms to deviate from their default value **f** to the deviant value **t**. If multiple causal mechanisms affect the same atom, the result is simple: there are no additive effects and the outcome is simply that the atom is **t** if and only if at least one mechanism causes it. If subsequent rules end up "causing" an effect that is already **t**, then this changes absolutely nothing.

It is to this setting that we now want to add negation-in-the-head. We will call such a negated literal in the head a *negative effect* literal. To be more precise, from now on, we allow rules of the form:

$$\forall \boldsymbol{x} \quad (L_1 : \alpha_1) \vee \cdots \vee (L_n : \alpha_n) \leftarrow \phi.$$

Here, ϕ is again a first-order logic formula with $\boldsymbol{x}$ as free variables and the $\alpha_i \in [0, 1]$ are again such that $\Sigma \alpha_i \leq 1$. Each of the L_i is now either a *positive effect literal* A (i.e., an atom) or a *negative effect literal* $\neg A$.

While the goal of this extension is of course to be able to represent such phenomena as the locking of the gear wheel described in the introduction, let us first take a step back and consider, in the abstract, which possible meanings this construct could reasonably have. Clearly, if for some atom A only positive effect literals are caused, the atom should end up being true, just as it always has. Similarly, if only negative effect literals $\neg A$ are caused, the atom A should be false. However, this does not even depend on the negative effect literals being present: because false is the default value in CP-logic, an atom will already be false whenever there are no positive effect literals for it, even if there are no negative effect literals either.

The only question, therefore, is what should happen if, for some A, both a positive and a negative effect literal are caused. One alternative could be that the result would somehow depend on the relative strength of the negative and positive effects, e.g., whether the power of aspirin to prevent a fever is "stronger" than the power of flu to cause it. However, such a semantics would be a considerable departure from the original version of CP-logic, in which cumulative effects are strictly ignored. In other words, CP-logic currently makes no distinction whatsoever between a headache that is simultaneously caused by five different conditions and a headache that has just a single cause. This design decision was made to avoid a logic that, in addition to probabilities, would also need to keep track of the degree to which a property holds. A logic combining probabilities with such fuzzy truth degrees would, in our opinion, become quite complex and hard to understand.

In this paper, we want to preserve the relative simplicity of CP-logic, and we will therefore again choose not to work with degrees of truth. Therefore, only two options remain: when both effect literals A and $\neg A$ are caused, the end result must be that A is either true of false. This basically means that, in the presence of both kinds of effect literals, we will have to choose to ignore one kind. It is obvious what this choice should be: the negative effect literals already have no impact on the semantics when there are only positive effect literals or when there are no positive effect literals, so if they would also have no impact when positive and negative effect literals are both present, then they would have never have any impact at all and we would have introduced a completely superfluous language construct. Therefore, the only reasonable choice is to give negative effect literals precedence over positive ones, that is, an atom A will be true if and only if it is caused at least once and no negative effect literal $\neg A$ is caused.

This can be formally defined by a minor change to the existing semantics of CP-logic. Recall that, in the current semantics, each node s of an execution model has an associated interpretation $\mathcal{I}(s)$, representing the current state of the world, and an associated three-valued interpretation $\mathcal{U}(s)$, representing an overestimate of all that could still be caused in s. We now add to this a third set, namely a set of atoms $\mathcal{N}(s)$, containing all atoms for which a negative effect literal has already been caused. The sets $\mathcal{I}(s)$ and $\mathcal{N}(s)$ evolve throughout an execution model as follows:

- In the root of the tree, $\mathcal{I}(s) = \mathcal{N}(s) = \{\}$
- When a *negative* effect literal $\neg A$ is caused in a node s, the execution model adds a child s' to s such that:
  - $\mathcal{N}(s') = \mathcal{N}(s) \cup \{A\}$;
  - $\mathcal{I}(s') = \mathcal{I}(s) \setminus \{A\}$.
- When a *positive* effect literal A is caused in a node s, the execution model adds a child s' to s such that:
  - $\mathcal{N}(s') = \mathcal{N}(s)$;
  - if $A \in \mathcal{N}(s)$, then $\mathcal{I}(s') = \mathcal{I}(s)$, else $\mathcal{I}(s') = \mathcal{I}(s) \cup \{A\}$.

Note that, throughout the execution model, we maintain the property that $\mathcal{N}(s) \cap \mathcal{I}(s) = \{\}$.

The overestimate $\mathcal{U}(s)$ is again constructed as the limit of a sequence of three-valued interpretations ν_i. To go from such a ν_i to ν_{i+1}, we make $\nu_{i+1}(A) = \mathbf{u}$ for all atoms A satisfying both of the following conditions:

- as before, $\nu_i(A) = \mathbf{f}$ and the positive effect literal A appears in the head of a rule $r \in \mathcal{R}(s)$ with $\nu_i(body(r)) \neq \mathbf{f}$;
- but now also $A \notin \mathcal{N}(s)$.

In this way, $\mathcal{U}(s)$ always assigns $\mathbf{t}$ to all atoms in $\mathcal{I}(s)$ and $\mathbf{f}$ to all those in $\mathcal{N}(s)$.

4 Encoding interventions

One of the interesting uses of negation-in-the-head is related to the concept of interventions, introduced by Pearl (2000). Let us briefly recall this notion. Pearl works in the context of *structural models*. Such a model is built from a number of random variables. For simplicity, we only consider boolean variables, i.e., atoms. These are again divided into exogenous and endogenous atoms. A structural model now consists of one equation $X := \varphi$ for each endogenous atom X, which defines that X is true if and only if the boolean formula φ holds. This set of equations should be acyclic (i.e., if we order the variables by defining that $X < Y$ if X appears in the equation defining Y, then this $<$ should be a strict order), in order to ensure that an assignment of values to the exogenous atoms induces a unique assignment of values to the endogenous ones.

A crucial property of causal models is that they can not only be used to predicts the normal behaviour of a system, but also to predict what would happen if outside factors unexpectedly intervene with its normal operation. For instance, consider the following simple model of which students must repeat a class:

$$Fail := \neg Smart \wedge \neg Effort. \qquad Repeat := Fail \wedge Required.$$

Under the normal operation of this "system", only students who are not smart can fail classes and be forced to repeat them. Suppose now that we catch a student cheating on an assignment and decide to fail him for the class. This action was not foreseen by the causal model, so it does not follow from the normal behaviour. In particular, failing the student may cause him to have to repeat the class, but if the student is actually smart, then failing him will not make him stupid. Pearl shows that we can model our action of failing the student by means of an *intervention*, denoted $do(Fail = \mathbf{t})$. This is a simple syntactic transformation, which removes and replaces the original equation for $Fail$:

$$Fail := \mathbf{t}. \qquad Repeat := Fail \wedge Required.$$

According to this updates set of equations, the student fails and may have to repeat the class, but he has not been made less smart.

In the context of CP-logic, let us consider the following simple medical theory:

$$(HighBloodPressure : 0.6) \leftarrow BadLifeStyle. \tag{14}$$

$$(HighBloodPressure : 0.9) \leftarrow Genetics. \tag{15}$$

$$(Fatigue : 0.3) \leftarrow HighBloodPressure. \tag{16}$$

Here, $BadLifeStyle$ and $Genetics$ are two exogenous predicates, which are both possible causes for $HighBloodPressure$. Suppose now that we observe a patient who suffers from $Fatigue$. Given our limited theory, this patient must be suffering from $HighBloodPressure$, caused by at least one of its two possible causes.

Now, suppose that a doctor is wondering whether it is a good idea to prescribe this patient some pills that cure high blood pressure. Again, the proper way to answer such a question is by means of an *intervention*, that first prevents the causal mechanisms that normally determine someone's blood pressure and then substitutes a new "mechanism" that just makes $HighBloodPressure$ false. This can be achieved by simply removing the two rules (14) and (15) from the theory. This is an instance of a general method, developed by Vennekens et al. (2010), of performing Pearl-style interventions in CP-logic. The result is that probability of $Fatigue$ drops to zero, i.e., $P(Fatigue \mid do(\neg HighBloodPressure)) = 0$.

In this way, we can evaluate the effect of prescribing the pills *without* actually having these pills in our model. This is a substantial difference to the way in which reasoning about actions is typically done in the field of knowledge representation, where formalisms such as situation or event calculus require an explicit enumeration of all available actions and their effects. Using an intervention, by contrast, we can envisage the effects of actions that we never even considered when writing our model.

Eventually, however, we may want to transform the above *descriptive* theory into a *prescriptive* one that tells doctors how to best treat a patient, given his or her symptoms. In this case, we would need rules such as this:

$$BPMedicine \leftarrow Fatigue. \tag{17}$$

Obviously, this requires us to introduce the action $BPMedicine$ of prescribing the medicine, which previously was implicit in our intervention, as an explicit action in our vocabulary. Negation-in-the-head allows us to syntactically express the effect of this new action: $\neg HighBloodPressure \leftarrow BPMedicine$.

This transformation can be applied in general, as the following theorem shows.

Theorem 1. *Let T be a CP-theory over a propositional vocabulary Σ. For an atom $A \in \Sigma$, let T' be the theory $T \cup \{r\}$ with r the rule $\neg A \leftarrow B$ and B an exogenous atom not in Σ. For each interpretation X for the exogenous atoms of T', if $B \in X$, then $\pi^X_{T'} = \pi^X_{do(T,\neg A)}$ and if $B \notin X$, then $\pi^X_{T'} = \pi^X_T$.*

This theorem shows that negation-in-the-head allows CP-theories to "internalize" the intervention of *doing* $\neg A$. The result is a theory T' in which the intervention can be switched on or off by simply choosing the appropriate interpretation for the exogenous predicate that now explicitly represents this intervention. Once the intervention has been syntactically added to the theory in this way, additional rules such as (17) may of course be added to turn it from an exogenous to an endogenous property.

It is important to note that this is a fully modular and elaboration tolerant encoding of the intervention, i.e., the original CP-theory is left untouched and the rules that describe the effect of the intervention-turned-action are simply added to it. This is something that we can only achieve using negation-in-the-head.

5 Representing defaults

An interesting test case for logic programs has always been the representation of defaults. The typical example concerns the default $\delta = \frac{Bird(x) : Flies(x)}{Flies(x)}$ together with the background knowledge: $\forall x\ Penguin(x) \Rightarrow \neg Flies(x)$. In an extended logic program, the two kinds of negation can be exploited to represent the default in an elegant way:

$$Flies(x) \leftarrow Bird(x) \wedge \texttt{not}\ \neg Flies(x). \qquad \neg Flies(x) \leftarrow Penguin(x).$$

In a normal logic program or deterministic CP-theory, defaults are typically represented using an *abnormality* predicate.

$$Flies(x) \leftarrow Bird(x) \wedge \neg Ab_\delta(x). \qquad Ab_\delta(x) \leftarrow Penguin(x).$$

Using CP-logic's new negation-in-the-head, the abnormality predicate can be omitted.

$$Flies(x) \leftarrow Bird(x). \tag{18}$$
$$\neg Flies(x) \leftarrow Penguin(x). \tag{19}$$

However, we do now lose the ability to distinguish between defeasible and non-defeasible rules, since negative effect literals can always be added to block any effect. In fact, this is necessary because of our desire to use negation-in-the-head to syntactically represent interventions (Section 4). It is after all a key property of Pearl's interventions that any causal relation in the model should, in principle, be open to intervention.

Even though, as this section shows, it is possible to use CP-logic to represent certain defaults, it is important to remember that it is not intended as a default logic. In particular, rule (18) should not actually be read as saying that birds normally fly. Instead, it says that, for each x, x being a bird causes it to be able to fly. Similarly, rule (19) says that being a penguin is a cause for being unable to fly. Note also that this is not a generally applicable methodology for representing defaults. For instance, if we wanted to state that penguins with jetpacks are an exception to rule (19), we would still have to introduce an abnormality predicate.

6 Probabilities and defaults

An interesting consequence of adding negation-in-the-head to CP-logic is that we can combine the encoding of defaults as in the previous section with uncertainty. For instance, let us suppose that there is, in general, a 5% change with which being a bird *does not* cause one to be able to fly. This may be the result, for instance, of a birth defect or some accident. This could be represented as follows:

$$(Flies(x) : 0.95) \leftarrow Bird(x). \tag{20}$$
$$\neg Flies(x) \leftarrow Penguin(x). \tag{21}$$

The first rule describes the normal situation for birds, whereas the second rule still serves to give an exception to the general rule. Note that, even for penguins, the causal mechanism underlying the first rule still happens, i.e., the rule is still fired, but it just fails to produce the outcomes of flying. Intuitively, we can think of this as the penguins still being born and being raised by their parents—i.e., they go through the same process of growing up that any bird goes through. It is just that, whereas this process causes the ability to fly for 95% of the normal birds, it never has this outcome for penguins. Of course, since learning to fly is actually the *only* possible effect of the first rule, the fact that this rule is still fired for penguins has no effect on anything.

The following example shows that this is not always the case.

$$(Wound(x) : 0.7) \vee (HoleInWall : 0.3) \leftarrow Shoot(x). \tag{22}$$

$$\neg Wound(x) \leftarrow Superhero(x). \tag{23}$$

Here, this first rule states that shooting a gun at someone might produce two possible effects: either the person ends up being wounded or the shot misses and causes instead a hole in the wall. The second rule adds an exception: if x happens to be a superhero, then x cannot be wounded. So, firing a gun at a superhero never causes $Wound(x)$, but with probability 0.3 still causes a hole in the wall.

This example also reveals a further way in which CP-logic is at heart a *causal* logic and not a logic of defaults. While we have so far been getting away with reading a rule such as (23) as expressing an exception to a default, this is not what it actually says: what this rule states is that being a superhero causes one to become "unwoundable". This does not only apply to wounds that would be caused by rule (22), but to all wounds. Therefore, if the CP-theory were to contain other causes for wounds, such as $(Wound(x) : 0.9) \leftarrow FallFromBuilding(x)$, then superheroes are automatically also protected against these.

7 Implementation

To implement the feature of negation-in-the-head, a simple transformation to regular CP-logic may be used. This transformation is based on the way in which Denecker and Ternovska (2007) encode causal ramifications in their inductive definition modelling of the situation calculus.

For a CP-theory T in vocabulary Σ, let $\Sigma_\neg$ consist of all atoms A for which a negative effect literal $\neg A$ appears in T. For each atom $A \in \Sigma_\neg$, we introduce two new atoms, C_A and $C_{\neg A}$. Intuitively, C_A means that there is a cause for A, and $C_{\neg A}$ means that there is a cause for $\neg A$. Let τ_A be the following transformation:

- Replace all positive effect literals A in the heads of rules by C_A
- Replace all negative effect literals $\neg A$ in the heads of rules by $C_{\neg A}$
- Add this rule: $A \leftarrow C_A \wedge \neg C_{\neg A}$

Let $\tau_\neg(T)$ denote the result of applying to T, in any order, all the transformations τ_A for which $A \in \Sigma_\neg$. It is clear that $\tau_\neg(T)$ is a regular CP-theory, i.e., one

without negation-in-the-head. As the following theorem shows, this reduction preserves the semantics of the theory.

Theorem 2. *For each interpretation X for the exogenous predicates, the projection of $\pi^X_{\tau_\neg(T)}$ onto the original vocabulary Σ of T is equal to π^X_T.*

When comparing the transformed theory $\pi_{\tau_\neg}(T)$ to the original theory T, we see that the main benefit of having negation-in-the-head lies in its *elaboration tolerance*: there is no need to know before-hand for which atoms we later might wish to add negative effect literals, since we can always add these later, without having to change to original rules. Both in the example of syntactically representing an intervention (Section 4) and that of representing exceptions to defaults (Section 5), this feature may be useful.

8 Conclusion

This paper is part of a long-term research project which aims to develop a Tarskian alternative to ASP: instead of relying on ASP's original epistemic intuitions, our goal is to have a language in which every expression can be interpreted as an objective statement about the real world. The first motivation for this is *simplicity*: many problems that are solved using present-day ASP systems and the GDT-methodology do not have an inherent epistemic component, so it would just be simpler if we could understand such programs in terms of what they say about the real world directly, instead of having to make a detour through the beliefs of some (irrelevant) rational agent. A second motivation is the *unity of science*: a huge effort has gone into both theoretical and practical research on classical logic. Its roots in Non-monotonic Reasoning have made ASP an antithesis to the classical approach, in which the desire to express objective knowledge is abandoned in favor of epistemic knowledge. Even though applications of ASP-solvers and SAT-solvers are often quite similar in practice, the "official" reading of ASP programs and classical theories is therefore radically different. The second goal is to bridge this gap.

An important part of this research project was the development of the language FO(ID), which showed how normal logic programs could be interpreted as *inductive definitions* and added in a meaningful way to classical logic. An extension of this work was the development of the language CP-logic, which allows non-deterministic and probabilistic causal processes to be expressed. In this paper, we have investigated the useful ASP feature of negation-in-the-head. We presented a meaningful interpretation of this feature in the context of CP-logic and discussed possibles uses of it. Finally, we also showed a simple transformation that reduces it to regular CP-logic.

References

C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming* 9(1):57-144, 2008.

M. Denecker and E. Ternovska. Inductive situation calculus. *Artificial Intelligence*, 171(5-6):332–360, 2007.

M. Denecker and J. Vennekens. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In *LPNMR*, volume 4483 of *LNCS*, pages 84–96. Springer, 2007.

M. Denecker, J. Vennekens, H. Vlaeminck, J. Wittocx, and M. Bruynooghe. Answer set programming's contributions to classical logic. An analysis of ASP methodology. In *MG-65: Symposium on Constructive Mathematics in Computer Science*, 2010.

M. Denecker, Y. Lierler, M. Truszczynski, and J. Vennekens. A tarskian informal semantics for asp. In *Technical Communications of the 28th International Conference on Logic Programming*, 2012.

M. Gelfond. Strong introspection. In *AAAI*, pages 386–391, 1991.

M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

N. Hall. Structural equations and causation. *Philosophical Studies*, 132(1): 109–136, 2007.

V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

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

C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13 (1):145–172, 1994.

J. Vennekens, M. Denecker, and M. Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009.

J. Vennekens, M. Denecker, and M. Bruynooghe. Embracing events in causal modelling: Interventions and counterfactuals in CP-logic. In *JELIA*, pages 313–325, 2010.