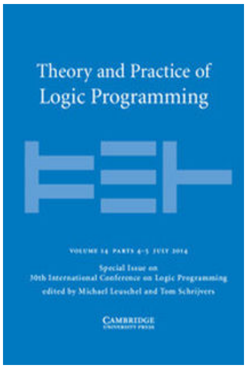


Theory and Practice of Logic Programming
VOLUME 14 PARTS 4-5 JULY 2014
Special Issue on 30th International Conference on Logic Programming
edited by Michael Leuschel and Tom Schrijvers
CAMBRIDGE UNIVERSITY PRESS

Causal Graph Justifications of Logic Programs

PEDRO CABALAR, JORGE FANDINNO and MICHAEL FINK

Theory and Practice of Logic Programming / Volume 14 / Special Issue 4-5 / July 2014, pp 603 - 618
DOI: 10.1017/S1471068414000234, Published online: 21 July 2014

TLP **14** (4–5): 603–618, 2014. © Cambridge University Press 2014
doi:10.1017/S1471068414000234

*Causal Graph Justifications of Logic Programs**

PEDRO CABALAR and JORGE FANDINNO
Department of Computer Science
University of Corunna, Spain
(*e-mail:* {`cabalar, jorge.fandino`}`@udc.es`)

MICHAEL FINK
Vienna University of Technology,
Institute for Information Systems
Vienna, Austria
(*e-mail:* `fink@kr.tuwien.ac.at`)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

In this work we propose a multi-valued extension of logic programs under the stable models semantics where each true atom in a model is associated with a set of justifications. These justifications are expressed in terms of *causal graphs* formed by rule labels and edges that represent their application ordering. For positive programs, we show that the causal justifications obtained for a given atom have a direct correspondence to (relevant) syntactic proofs of that atom using the program rules involved in the graphs. The most interesting contribution is that this causal information is obtained in a purely semantic way, by algebraic operations (product, sum and application) on a lattice of causal values whose ordering relation expresses when a justification is stronger than another. Finally, for programs with negation, we define the concept of *causal stable model* by introducing an analogous transformation to Gelfond and Lifschitz's program reduct. As a result, default negation behaves as "absence of proof" and no justification is derived from negative literals, something that turns out convenient for elaboration tolerance, as we explain with a running example.

KEYWORDS: Answer Set Programming, Causality, Knowledge Representation, Multi-valued Logic Programming

1 Introduction

An important difference between classical models and most Logic Programming (LP) semantics is that, in the latter, true atoms must be founded or justified by a given derivation. Consequently, falsity is understood as absence of proof: for instance, a common informal way reading for default literal *not p* is "there is no way to derive *p*." Although this idea seems quite intuitive, it actually resorts to

* This research was partially supported by Spanish MEC project TIN2009-14562-C05-04, Xunta projects GPC2013/070 and INCITE 2011, Inditex-University of Corunna 2013 grants and the Austrian Science Fund (FWF) project P24090.

a concept, the *ways to derive p*, outside the scope of the standard LP semantics. In other words, LP semantics point out whether there exists some derivation for an atom, but do not provide the derivations themselves, if several alternatives exist.

However, such information on justifications for atoms can be of great interest for Knowledge Representation (KR), and especially, for dealing with problems related to causality. In the area of diagnosis, for instance, when a discrepancy between expected and observed behaviour is found, it may be convenient to not only exhibit a set of malfunctioning components as explanation, but also the way (a causal graph) in which these breakdowns have eventually caused the discrepancies. Another potential application area is legal reasoning where determining a legal responsability usually involves finding out which agent or agents have eventually caused a given result, regardless the chain of effects involved in the process. An important challenge in causal reasoning is the capability of not only deriving facts of the form "A has caused B," but also being able to represent them and reason about them. As an example, take the assertion:

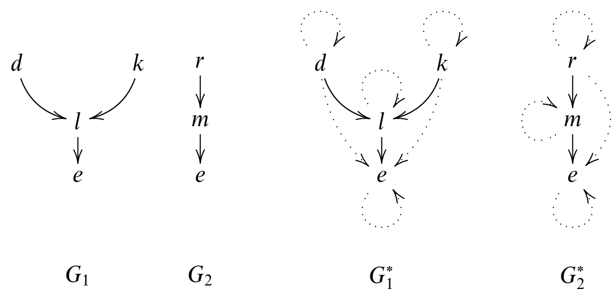
"If somebody causes an accident, (s)he is legally responsible for that."

This law does not specify the possible ways in which a person may cause an accident. Depending on a representation of the domain, the chain of events from the agent's action(s) to the final effect may be simple (a direct effect) or involve a complex set of indirect effects and defaults like inertia. Regarding representation of the above law, for instance, one might think of an informal rule:

$$\mathit{responsible}(X,Y) \leftarrow \mathit{action}(A),\ \mathit{person}(X),\ \mathit{accident}(Y),\ \text{“}\mathit{do}(A,X)\ \texttt{caused}\ \mathit{occurs}(Y)\text{”}\ .$$

If the pseudo-literal "$\mathit{do}(A,X)$ `caused` $\mathit{occurs}(Y)$" actually corresponds to an explicit representation of all the possible ways of causing an accident, however, one immediately runs into a problem of *elaboration tolerance* (McCarthy 1998) — adding new rules that causally connect $\mathit{do}(A,X)$ to $\mathit{occurs}(Y)$ (in a direct or indirect way) would force us to build new rules for $\mathit{responsible}(X,Y)$. What is needed instead, and what we actually propose as an eventual aim and future extension of our work, is to introduce, indeed, some kind of new LP literal "A caused B," with *an associated semantics* capable of revealing causes A of a given true atom B.

While not straightforward, the rewarding perspective of such a semantic approach is an extension of Answer Set Programming (ASP) (Brewka *et al.* 2011) with causal literals capable of representing different kinds of causal influences (sufficient cause, necessary cause, etc). In this paper, we tackle the above issue and, as a first step and basic underlying requirement, develop a suitable semantics capable of associating causal justifications with each true atom. To this end, we propose a multi-valued extension of logic programs under the stable model semantics (Gelfond and Lifschitz 1988) where each true atom in a model is associated with a set of justifications in the form of *causal graphs*. To further illustrate our motivation, consider the following example.



(a) Causal graphs (b) Reflexive, transitive closures

Fig. 1. Derivations G_1 and G_2 justifying atom *prison* in program P_1.

Example 1 (*From Cabalar 2011*)
Some country has a law l that asserts that driving drunk is punishable with imprisonment. On the other hand, a second law m specifies that resisting arrest has the same effect. The execution e of a sentence establishes that a punishment implies imprisonment. Suppose that some person drove drunk and resisted to be arrested. □

We can capture this scenario with the following logic program P_1:

$$\begin{array}{lll} l : punish \leftarrow drive,\ drunk & m : punish \leftarrow resist & e : prison \leftarrow punish \\ d : drive & k : drunk & r : resist \end{array}$$

The least model of this positive program makes atom *prison* true, so we know that there exists a possible derivation for it. In particular, two alternative justifications can be made, corresponding to the graphs in Figure 1(a): driving drunk and, independently, resisting to authority (vertices and edges respectively corresponds with rule labels and their dependences).

More specifically, we summarise our contributions as follows.

- We define a multi-valued semantics for (normal) logic programs based on causal graphs. An important result is that, despite of this semantic nature, we are able to show that causal values have a direct correspondence to (relevant) syntactic proofs using the program rules involved in the graphs (cf. Section 4).
- We also define an ordering relation that specifies when a cause is *stronger* than another, and show how causal values form a lattice with three associated algebraic operations: a product '$*$' representing conjunction or joint causation; a sum '$+$' representing alternative causes; and a non-commutative product '$\cdot$' that stands for rule application. We study beneficial properties of these operations that allow manipulating and reasoning with causal values in an analytical way (cf. Sections 2 and 3).

Fostered by its algebraic treatment of causal values, our work facilitates the incorporation of dedicated, more specific causal expressions representing causal influence of different kinds.

2 Causes as graphs

In this and subsequent Section 3, we introduce the lattice of causal values in two different steps. In a first step, we focus on the idea of an "individual" cause and then we proceed to explain the concept of causal value that allows collecting different alternative causes.

We begin recalling several graph definitions and notation. A (directed) *graph* is a pair $\langle V, E \rangle$ where V is a set of vertices V and E is a set of edges $E \subseteq V \times V$. In the following definitions, let $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ be two graphs. We say that G is a *subgraph* of G', written $G \subseteq G'$, when $V \subseteq V'$ and $E \subseteq E'$. We write $G \cup G'$ to stand for the graph $\langle V \cup V', E \cup E' \rangle$. We represent the reflexive and transitive closure of G as G^*. Finally, we introduce a *concatenation* operation $G \odot G'$ on graphs corresponding to a graph with vertices $V \cup V'$ and edges $E \cup E' \cup \{(x, y) \mid x \in V, y \in V'\}$. Notice that, $G \cup G' \subseteq G \odot G'$, that is, the concatenation extends the union of graphs by adding all possible arcs that go from some node in G to some node in G'.

Definition 1 (*Causal graph*)
Given some set Lb of (rule) labels, a *causal graph* (*c-graph*) G is a reflexively and transitively closed directed graph, i.e., $G^* = G$, whose vertices are labels, i.e. $V \subseteq Lb$. We denote by $\mathbf{C}_{Lb}$ the set of all possible causal graphs over Lb. □

Intuitively, the vertices correspond to rules involved in a derivation of a given atom (or formula), and the edges point out a (partial) ordering of application of rules in the derivation. Figure 1(a) shows two causal graphs with labels from P_1. Transitivity is crucial for interpreting the subgraph relation $G \subseteq G'$ as a way to express that G' is redundant with respect to G. For instance, a graph G_3 formed by the single edge (r, e) *is not a subgraph* of G_2 but is simpler (does not require using m). This fact is captured by $G_3^* \subseteq G_2^*$. Reflexivity is convenient for simpler definitions. For instance, the causal graph formed by a single label l also has a single edge (l, l)—we call this an *atomic* causal graphs and represent it just by its label. For simplicity, we will usually omit transitive and reflexive arcs when depicting a causal graph. For instance, taking G_1 and G_2 in Figure 1(a) as causal graphs actually amounts to considering the graphs shown in Figure 1(b), where previously omitted arcs are shown as dotted lines. We define next a natural ordering relation among them.

Definition 2 (*Sufficient*)
A causal graph G is *sufficient* for another causal graph G', written $G \leqslant G'$, when $G \supseteq G'$. □

Saying that G is sufficient for G' intuitively means that G contains enough information to yield the same effect than G', but perhaps more than needed (this explains $G \supseteq G'$). For this reason, we sometimes read $G \leqslant G'$ as "G' is *stronger* than G."

Since graphs with the subgraph relation form a poset, the set of causal graphs also constitutes a poset $\langle \mathbf{C}_{Lb}, \leqslant \rangle$ with a top element corresponding to the empty c-graph $G_\emptyset = \langle \emptyset, \emptyset \rangle$. This stands for a kind of "absolute truth" and is of interest for including rules or facts one does not want to label, that is, their effect will not be traced in the justifications.

Any causal graph can be built up from labels (atomic causal graphs) using two basic operations: the product $G * G' \overset{\text{def}}{=} (G \cup G')^*$ that stands for union of causal graphs or *joint interaction*, and the concatenation $G \cdot G' \overset{\text{def}}{=} (G \odot G')^*$ that captures their *sequential application*. The reason for applying a closure is that the result of $G \cup G'$ and $G \odot G'$ does not need to be closed under transitivity. We can extend the product to any (possibly empty and possibly infinite) set of causal graphs S so that $\prod S \overset{\text{def}}{=} \left(\bigcup_{G \in S} G \right)^*$.

Example 2 (*Ex. 1 continued*)
The cause for the body of l in P_1 is the product of causes for *drive* and *drunk*, that is $d * k$ formed with vertices $\{d, k\}$ and edges $\{(d, d), (k, k)\}$. As a result, the explanation of the rule head, *punish*, is formed by the concatenation of its body cause $d * k$ with its label, that is $(d * k) \cdot l$. In its turn, this becomes the cause for the body of e and so, we get the explanation $(d * k) \cdot l \cdot e$ for atom *prison* represented as G_1 in Figure 1(b). Similarly, G_2 corresponds to $r \cdot m \cdot e$. □

When writing these causal expressions, we assume that '$\cdot$' has higher priority than '$*$'. Furthermore, we will usually omit '$\cdot$' when applied to consecutive labels, so that $r \cdot m \cdot e$ will be abbreviated as *rme*. It is easy to see that $G * G' = G' * G$ while, in general, $G \cdot G' \neq G' \cdot G$, that is, concatenation is not commutative. Another observation is that $G \cdot G' \leqslant G * G'$, that is, concatenation is sufficient for the product, but the opposite does not hold in general. Moreover, in our running example, we can check that $(d * k) \cdot l$ is equal to $(d \cdot l) * (k \cdot l)$. In fact, application distributes over products and, as a result, we can identify any causal graph with the product of all its edges. To conclude this section, we note that the set of causal graphs $\mathbf{C}_{Lb}$ ordered by $\leqslant$ forms a lower semilattice $\langle \mathbf{C}_{Lb}, * \rangle$, where the product constitutes the infimum.

3 Alternative causes

Having settled the case for individual causes, let us now proceed to represent situations in which several alternative and independent causes can be found for an atom p. The obvious possibility is just using a *set of causal graphs* for that purpose. However, we should additionally disregard causes for which a stronger alternative exists. For instance, as we saw before, cause *rme* is sufficient for explaining *punish* and therefore, it is also an alternative way to prove this atom, but redundant in the presence of the stronger cause *rm*. This suggests to choose sets of $\leqslant$-maximal causal graphs as 'truth values' for our multi-valued semantics. In principle, this is the central idea, although $\leqslant$-maximal causal graphs incur some minor inconveniences in mathematical treatment. For instance, if we collect different alternative causes by just using the union of sets of maximal causal graphs, the elements in the result need not be maximal. Besides, the operations of product and concatenation are expected to extend to the sets adopted as causal values. To address these issues, a more solid representation is obtained resorting to *ideals* of causal graphs.

Given any poset $\langle A, \leqslant \rangle$, an *ideal* I is any set $I \subseteq A$ satisfying[1]: if $x \in I$ and $y \leqslant x$ then $y \in I$. A compact way of representing an ideal I is by using its set of maximal elements S, since the rest of I contains *all elements below* them. The *principal ideal* of an individual element $x \in A$ is denoted as $\downarrow x \overset{\text{def}}{=} \{y \in A \mid y \leqslant x\}$. We extend this notion for any set of elements S so that $\downarrow S \overset{\text{def}}{=} \bigcup\{\downarrow x \mid x \in S\} = \{y \in A \mid y \leqslant x, \text{for some } x \in S\}$. Thus, we will usually represent an ideal I as $\downarrow S$ where S are the maximal [2] elements in I. In fact, maximal elements constitute the relevant information provided by the ideal, while keeping all other elements is convenient for simplicity of algebraic treament (but we do not assign a particular meaning to them).

Definition 3 (*Causal Value*)
Given a set of labels Lb, a *causal value* is any ideal for the poset $\langle \mathbf{C}_{Lb}, \leqslant \rangle$. We denote by $\mathbf{V}_{Lb}$ the set of causal values. □

Product, concatenation and the $\leqslant$-relation are easily extended to any pair U, U' of causal values respectively as: $U * U' \overset{\text{def}}{=} U \cap U'$ and $U \cdot U' \overset{\text{def}}{=} \downarrow \{\ G \cdot G' \mid G \in U \text{ and } G' \in U'\ \}$ and $U \leqslant U'$ iff $U \subseteq U'$. We also define addition as: $U + U' \overset{\text{def}}{=} U \cup U'$ allowing to collect alternative causes. Using terminology and results from lattice theory in (Stumme 1997) we can prove the following.

Theorem 1
$\mathbf{V}_{Lb}$ forms a free, completely distributive lattice with join $+$ and meet $*$ generated by the lower semilattice $\langle \mathbf{C}_{Lb}, * \rangle$ with the injective homomorphism (or embedding) $\downarrow : \mathbf{C}_{Lb} \longrightarrow \mathbf{V}_{Lb}$. □

Essentially, this theorem means that the mapping $\downarrow$ from c-graph G to its principal ideal $\downarrow G$ is preserved for their respective products, $\downarrow(G_1 * G_2) = \downarrow G_1 * \downarrow G_2$, and ordering relations: $G_1 \leqslant G_2$ (among c-graphs) iff $\downarrow G_1 \leqslant \downarrow G_2$ (among causal values).

Example 3 (*Ex. 1 continued*)
The interpretation for *punish* has two alternative causes $(d * k) \cdot l$ and rm that become the causal values $\downarrow(d * k) \cdot l$ and $\downarrow rm$. The causal value for *punish* is then formed by their addition:

$$\downarrow(d * k) \cdot l \ + \ \downarrow rm \ = \ \downarrow(d * k) \cdot l \ \cup \ \downarrow rm \ = \ \downarrow\{\ (d * k) \cdot l,\ rm\ \}$$

This ideal contains, among others, the cause rme, although it is not maximal due to rm:

$$\downarrow\{\ (d * k) \cdot l,\ rm\ \} \ \cup \ \downarrow rme \ = \downarrow\{\ (d * k) \cdot l,\ rm,\ rme\} \ = \downarrow\{\ (d * k) \cdot l,\ rm\ \} \quad \square$$

[1] We use terminology from (Stumme 1997). In some texts this is known as *semi-ideal* or *down-set* to differentiate this definition from the stronger case in which ideals are applied on a (full) lattice rather than a semi-lattice.

[2] Note that, in the case of causal graphs, the existence of maximal elements for the $\leqslant$-relation amounts to the existence of minimal elements for the subgraph relation, and this holds since the latter is well-founded.

Absorption	*Associativity*	*Identity*	*Annihilator*
$t \quad = t + u \cdot t \cdot w$	$t \cdot (u{\cdot}w) = (t{\cdot}u) \cdot w$	$t = 1 \cdot t$	$0 = t \cdot 0$
$u \cdot t \cdot w = t * u \cdot t \cdot w$		$t = t \cdot 1$	$0 = 0 \cdot t$

Addition distributivity	*Product distributivity*
$t \cdot (u{+}w) = (t{\cdot}u) + (t{\cdot}w)$	$c \cdot d \cdot e = (c \cdot d) * (d \cdot e)$ with $d \neq 1$
$(t + u) \cdot w = (t{\cdot}w) + (u{\cdot}w)$	$c \cdot (d * e) = (c \cdot d) * (c \cdot e)$
	$(c * d) \cdot e = (c \cdot e) * (d \cdot e)$

Fig. 2. Properties of the '·' operator (c, d, e are terms without '+').

The term completely distributive lattice in Theorem 1 means that meet (resp. join) operation is defined for any infinite subset of $\mathbf{V}_{Lb}$ and distributes over infinite joins (resp. meets). There is also a bottom element, the empty ideal $\emptyset$ (standing for "falsity") that will be denoted as 0, and a top element, the ideal formed by the empty causal graph $\downarrow G_{\emptyset} = \mathbf{C}_{Lb}$ (standing for "absolute truth") that is denoted as 1 from now on. To improve readability, we introduce the syntactic notion of *causal terms*, that allow representing the possible causal values without explicitly resorting to graphs or ideals.

Definition 4 (*Causal term*)

A *(causal) term*, t, over a set of labels Lb, is recursively defined as one of the following expressions $t ::= l \mid \prod S \mid \sum S \mid t_1 \cdot t_2$ where $l \in Lb$, t_1, t_2 are in their turn causal terms and S is a (possibly empty and possible infinite) set of causal terms. When S is finite and non-empty, $S = \{t_1, \ldots, t_n\}$ we write $\prod S$ simply as $t_1 * \cdots * t_n$ and $\sum S$ as $t_1 + \cdots + t_n$. □

We assume that '*' has higher priority than '+'. The causal value associated to a causal term is naturally obtained by the recursive application of its operators until we reach the level of labels, so that each label l in the term actually stands for the principal ideal $\downarrow l$. When $S = \emptyset$, the union in S corresponds to the bottom causal value that is, $\sum \emptyset = 0$. Analogously, the intersection of elements in $S = \emptyset$ corresponds to top causal value, i.e., $\prod \emptyset = 1$.

From now on, we will use causal terms as compact representations of causal values. Individual causes (i.e. causal graphs) correspond to terms without addition (note that this also excludes 0, the empty sum). Several interesting algebraic properties can be proved for causal values. In particular, Theorem 1 guarantees that they form a free completely distributive lattice with respect to '*' and '+' satisfying the standard properties such as associativity, commutativity, idempotence, absorption or distributivity on both directions[3]. Besides, as usual, 0 (resp. 1) is the annihilator for '*' (resp. '+') and the identity for '+' (resp. '*'). More significantly, the main properties for '·' are shown in Figure 2.

[3] The term "free lattice" in Theorem 1 means that *any* equivalence with * and + can be derived from these properties.

4 Positive programs and minimal models

Let us now reconsider logic programs and provide a semantics based on the causal values we have just defined. For the syntax, we recall standard LP definitions, just slightly extending it by introducing rule labels. A *signature* is a pair $\langle At, Lb\rangle$ of sets that respectively represent a set of *atoms* (or *propositions*) and a set of *labels.* As usual, a *literal* is defined as an atom p (positive literal) or its default negation $not\ p$ (negative literal). In this paper, we will concentrate on programs without disjunction in the head (leaving its treatment for future work).

Definition 5 (*Causal logic program*)
Given a signature $\langle At, Lb\rangle$, a *(causal) logic program P* is a (possible infinite) set of rules of the form:

$$t : H \leftarrow B_1, \dots, B_n, \tag{1}$$

where $t \in Lb \cup \{1\}$, H is an atom (the *head* of the rule) and $B_1, \dots, B_n$ are literals (the *body*). □

For any rule R of the form (1) we define $label(R) \stackrel{\text{def}}{=} t$. We denote by $head(R) \stackrel{\text{def}}{=} H$ its *head*, and by $body(R) \stackrel{\text{def}}{=} \{B_1, \dots, B_n\}$ its *body*. When $n = 0$ we say that the rule is a *fact* and omit the symbol '$\leftarrow$.' When $t \in Lb$ we say that the rule is labelled; otherwise $t = 1$ and we omit both t and ':'. By these conventions, for instance, an unlabelled fact p is actually an abbreviation of $(1 : p \leftarrow)$. A logic program P is *positive* if it contains no default negation. A program is *uniquely labelled* if no pair of labelled rules share the same label, and *completely labelled* if, additionally, all rules are labelled. For instance, P_1 is completely labelled.

Given a signature $\langle At, Lb\rangle$ a *causal interpretation* is a mapping $I : At \longrightarrow \mathbf{V}_{Lb}$ assigning a causal value to each atom. For any interpretations I, J, we say that $I \leqslant J$ when $I(p) \leqslant J(p)$ for each atom $p \in At$. Hence, there is a $\leqslant$-bottom (resp. $\leqslant$-top) interpretation $\mathbf{0}$ (resp. $\mathbf{1}$) that stands for the interpretation mapping each atom p to 0 (resp. 1). The value assigned to a negative literal $not\ p$ by an interpretation I, denoted as $I(not\ p)$, is defined as: $I(not\ p) \stackrel{\text{def}}{=} 1$ if $I(p) = 0$; and $I(not\ p) \stackrel{\text{def}}{=} 0$ otherwise. An interpretation is *two-valued* if it maps all atoms to $\{0, 1\}$. Furthermore, for any causal interpretation, its corresponding two-valued interpretation, written I^{cl}, is defined so that for any atom p: $I^{cl}(p) \stackrel{\text{def}}{=} 0$ if $I(p) = 0$; and $I^{cl}(p) \stackrel{\text{def}}{=} 1$ otherwise.

Definition 6 (*Causal model*)
Given a positive causal logic program P, a causal interpretation I is a *causal model*, in symbols $I \models P$, if and only if, for each rule $R \in P$ of the form (1), the following condition holds:

$$\big(\ I(B_1) * \dots * I(B_n)\ \big) \cdot t \leqslant I(H)$$ □

Example 4 (*Ex. 1 continued*)
Take rule l from program P_1 and let I be such that $I(drive) = d$ and $I(drunk) = k$. Then I will be a model of l when $(d * k) \cdot l \leqslant I(punish)$. In particular, this holds when $I(punish) = (d * k) \cdot l + r \cdot m$ which was the value we expected for that atom. But it would also hold when, for instance, $I(punish) = l + m$ or $I(punish) = 1$.

The inequality in Definition 6 is important to accommodate possible additional facts such as $(l : punish)$ or even $(1 : punish)$ in the program. □

The fact that any $I(punish)$ greater than $(d * k) \cdot l + r \cdot m$ also becomes a model clearly points out the need for selecting *minimal* models. In fact, as it is the case for non-causal programs, positive programs have a $\leqslant$-least model that can be computed by iterating an extension of the well-known *direct consequences operator* (van Emden and Kowalski 1976).

Definition 7 (*Direct consequences*)
Given a positive logic program P over signature $\langle At, Lb \rangle$, the operator of *direct consequences* is a function $T_P : \mathbf{I} \longrightarrow \mathbf{I}$ such that, for any causal interpretation I and any atom $p \in At$:

$$T_P(I)(p) \overset{\text{def}}{=} \sum \{ \; \big(\, I(B_1) * \ldots * I(B_n) \, \big) \cdot t \;\; | \;\; (t : p \leftarrow B_1, \ldots, B_n) \in P \; \}$$

Theorem 2
Let P be a (possibly infinite) positive logic program with n causal rules. Then, (*i*) $lfp(T_P)$ is the least model of P, and (*ii*) $lfp(T_P) = T_P \uparrow^{\omega} (\mathbf{0}) = T_P \uparrow^{n} (\mathbf{0})$. □

The proof of this theorem relies on an encoding of causal logic programs into *Generalized Annotated Logic Programming* (GAP) (Kifer and Subrahmanian 1992) and applying existing results for that general multi-valued LP framework. Theorem 2 just guarantees that the least fixpoint of T_P is well-behaved, but does not explain the nature of the obtained causal values. We illustrate next that these values have a direct relation to the syntactic idea of *proof* in a positive program.

Definition 8
Given a positive program P, a *proof* $\pi(p)$ of an atom p can be recursively defined as a derivation:

$$\pi(p) \overset{\text{def}}{=} \frac{\pi(B_1) \; \ldots \; \pi(B_n)}{p} \; (R),$$

where $R \in P$ is a rule with $head(R) = p$ and $body(R) = \{B_1, \ldots, B_n\}$. When $n = 0$, the derivation antecedent $\pi(B_1) \; \ldots \; \pi(B_n)$ is replaced by $\top$ (corresponding to the empty body). □

Each derivation in a proof is a particular application of Modus Ponens where, once the body (conjunction of literals B) of a rule R $(p \leftarrow B)$ has been proved, then the head p can be concluded.

Example 5 (*Ex. 1 continued*)
Program P_1 is positive and, in fact, completely labelled, so we can identify each rule with its label. Atom *prison* can be derived in P_1 using the two proofs on the left in Figure 3. These two proofs have a clear correspondence to causes $(d * k) \cdot le$ and rme depicted in Figure 1(b). In fact, the least model I of P_1 assigns causal value $I(punish) = (d * k) \cdot le + rme$. □

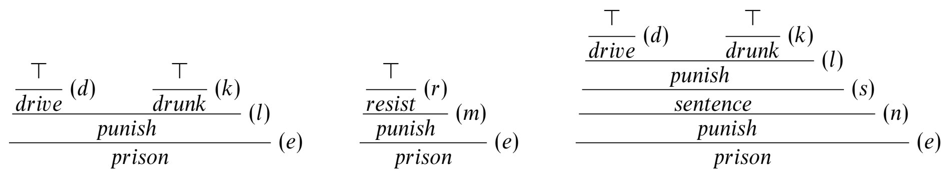


Fig. 3. Some proofs for atom *prison* (the rightmost proof is redundant).

Let P be a positive, completely labelled program. Given a proof π, we define its graph G_π as follows. For each sub-derivation in π of the form $\pi(p)$ in Definition 8 we include an edge (l_i, m) where m is the label of rule R and l_i is the label of the top-level rule in $\pi(B_i)$, for all $i = 1, \ldots, n$. The vertices in G_π exclusively collect the labels in those edges. We define $graph(\pi) \stackrel{\text{def}}{=} G^*_\pi$. The two left proofs in Figure 3 are then obviously mapped to the causal graphs in Figure 1(b). If Π is a set of proofs, we define $graph(\Pi) \stackrel{\text{def}}{=} \{graph(\pi) \mid \pi \in \Pi\}$.

A proof can be sometimes redundant, in the sense that some of its derivation steps could be removed. A natural way of defining a non-redundant proof is resorting to its associated graph. We say that a proof $\pi(p)$ of an atom p in a positive, completely labelled program P is *redundant* if there exists another proof of p, $\pi'(p)$, such that $graph(\pi(p)) \leqslant graph(\pi'(p))$, in other words, we can build another proof π' with a smaller associated graph.

Example 6

Suppose that we introduce an atom *sentence* which acts as a synonym for *punish*. Furthermore, assume law m mentions *sentence* as its head now, instead of *punish*. Hence, let P_2 be program:

$$\begin{array}{lll} l : punish \leftarrow drive, drunk & d : drive & n : punish \leftarrow sentence \\ m : sentence \leftarrow resist & k : drunk & s : sentence \leftarrow punish \\ e : prison \leftarrow punish & r : resist & \end{array}$$

Then, the rightmost proof shown in Figure 3 together with its associated graph $(d * k) \cdot lsne$ is redundant, since the (still valid) leftmost proof in Figure 3 for *prison* has an associated stronger cause (or smaller graph) $(d*k) \cdot le$. Considering the positive loop formed by n and s, one may wonder why it does not spoil the computation of T_{P_2} to iterate forever (adding more and more concatenations of n and s). The reason is that, at a given point, subsequent iterations yield redundant graphs subsumed by previous steps. In particular, the iteration of T_{P_2} yields the steps:

i	*drive*	*drunk*	*resist*	*sentence*	*punish*	*prison*
1	d	k	r	0	0	0
2	d	k	r	rm	$(d * k) \cdot l$	0
3	d	k	r	$rm + (d * k) \cdot ls$	$(d * k) \cdot l + rmn$	$(d * k) \cdot le$
4	d	k	r	$rm + (d * k) \cdot ls$	$(d * k) \cdot l + rmn$	$((d * k) \cdot l + rmn) \cdot e$

reaching a fixpoint at step 4. The value for *sentence* at step 4 would actually be the sum of rm (derived from *resist*) with the value of *punish* in the previous step,

$(d * k) \cdot l + rmn$ followed by s. This corresponds to:

$$\begin{aligned} rm + (\underbrace{(d * k) \cdot l + rmn}_{punish}) \cdot s &= rm + (d * k) \cdot ls + rmns && \text{distributivity} \\ &= rm + rmns + (d * k) \cdot ls && \text{commutativity} \\ &= rm + rm \cdot ns + (d * k) \cdot ls && \text{associativity} \\ &= rm + 1 \cdot rm \cdot ns + (d * k) \cdot ls && \text{identity} \\ &= rm + (d * k) \cdot ls && \text{absorption for '+' and '}\cdot\text{'} \end{aligned}$$

That is, iterating the loop *rmns* is redundant since a stronger cause *rm* was obtained before. □

Theorem 3
Let P be a positive, completely labelled program, and Π_p the set of non-redundant proofs of some atom p with respect to P. If I denotes the least model of P, then:

$$G \in graph(\Pi_p) \quad \text{iff} \quad G \text{ is a maximal causal graph in } I(p) \qquad \Box$$

Note the importance of this result: it reveals that the information we obtain by a purely semantic treatment of causal values (computing the least model by algebraic operations) has a one-to-one correspondence to syntactic proofs obtained by modus ponens that are further guaranteed to be non-redundant (they do not contain unnecessary steps). Completely labelled programs are interesting for establishing the correspondence in the theorem above, but there are several scenarios in which one may be interested in disregarding the effect of rules in a program or in identifying a group of rules under the same label.

Example 7
Let P_3 be the following variation of P_2:

$$\begin{array}{lll} z : sentence \leftarrow drive, drunk & d : drive & punish \leftarrow sentence \\ z : punish \leftarrow resist & k : drunk & sentence \leftarrow punish \\ e : prison \leftarrow punish & r : resist & \end{array}$$

where l and m in P_2 are now just two cases of a common law z, and *punish* and *sentence* depend on each other through unlabelled rules. □

Removing the labels in the positive cycle between *sentence* and *punish* captures the idea that, since they are synonyms, whenever we have a cause for *sentence*, it immediately becomes a cause for *punish* and vice versa. By iterating the T_P operator, it is not difficult to see that the least causal model I_3 makes the assignments $I_3(sentence) = I_3(punish) = (d * k) \cdot z + rz$ (that is *sentence* and *punish* are equivalent) and $I_3(prison) = (d * k) \cdot ze + rze$. This result could also be computed from the least model I_2 for P_2 by replacing l and m by z and "removing" n and s (that is, replacing them by 1). This is, in fact, a general property we formalise as follows. Given two causal terms t, u and a label l, we define $t[l \mapsto u]$ as the result of replacing label l in t by term u.

Theorem 4
Let P be a positive causal logic program and P' be the result of replacing a label l in P by some u, where u is any label or 1. Furthermore, let I and I' be the least models of P and P', respectively. Then, $I'(p) = I(p)[l \mapsto u]$ for any atom p. □

In particular, in our example, $I_3(p) = I_2(p)[l \mapsto z][m \mapsto z][n \mapsto 1][s \mapsto 1]$, for any atom p. If we remove all labels in a program, we eventually get a standard, unlabelled program. Obviously, its least model will be two-valued, since removing all labels in causal terms, eventually collapses all of them to $\{0, 1\}$. As a result, we can easily establish the following correspondence.

Theorem 5
Let P be a causal positive logic program and P' its unlabelled version. Furthermore, let I be the least causal model of P and I' the least classical model of P'. Then $I' = I^{cl}$. □

5 Default negation

To introduce default negation, let us consider the following variation of our running example.

Example 8
Assume now that law e is a default and that there may be exceptional cases in which punishment is not effective. In particular, some of such exceptions are a pardon, that the punishment was revoked, or that the person has diplomatic immunity. A possible program P_4 encoding this variant of the scenario is:

$$\begin{array}{rlll} l & : punish \leftarrow drive, drunk & d : drive & abnormal \leftarrow pardon \\ m & : punish \leftarrow resist & k : drunk & abnormal \leftarrow revoke \\ e & : prison \leftarrow punish, not\ abnormal & r : resist & abnormal \leftarrow diplomat \end{array}$$

This program has a unique stable model which still keeps *prison* true, since *no proof for abnormal* could be obtained, i.e. no exception occurred. □

From a causal perspective, saying that the lack of an exception is part of a cause (e.g., for imprisonment) is rather counterintuitive. It is not the case that we go to prison because of not receiving a pardon, not having a punishment revocation, not being a diplomat, or whatever possible exception that might be added in the future[4]. Instead, as nothing violated default e, the justifications for *prison* should be those shown in Figure 1(a). In this way, falsity becomes the *default situation* that is broken when a cause is found[5]. This interpretation carries over to negative literals, so that the presence of *not p* in a rule body does not propagate causal information, but instead is a check for the absence of an exception. To capture this behaviour, we

[4] A case of the well-known *qualification problem* (McCarthy 1977), i.e., the impossibility of listing all the possible conditions that prevent an action to cause a given effect. Appendix B (available online) contains a more elaborated example showing how the qualification problem may affect causal explanations when inertia is involved.

[5] The paper (Hitchcock and Knobe 2009) contains an extended discussion with several examples showing how people ordinarily understand causes as deviations from a norm.

proceed to extend the traditional program reduct (Gelfond and Lifschitz 1988) to causal logic programs.

Definition 9 (*Program reduct*)
The *reduct* of program P with respect to causal interpretation I, in symbols P^I, is the result of:

1. removing from P all rules R, s.t. $I(B) \neq 0$ for some negative literal $B \in body(R)$;
2. removing all negative literals from the remaining rules of P. □

An interpretation I is a *causal stable model* of program P iff I is the least causal model of P^I.

Example 9 (*Ex. 8 continued*)
Suppose that we add atoms $(p : pardon)$ and $(d : diplomat)$ to program P_4. The only stable model I of this extended program makes $I(prison) = 0$ and $I(abnormal) = p{+}d$ as expected. □

Theorem 6 (*Correspondence to non-causal stable models*)
Let P be a causal logic program and P' its unlabelled version. Then:

1. If I is a causal stable model of P, then I^{cl} is a stable model of P'.
2. If I' is a stable model of P' then there is a unique causal stable model I of P s.t. $I' = I^{cl}$. □

This theorem also shows a possible method for computing causal stable models of a program P. We may first run a standard ASP solver on the unlabelled version of P to obtain a stable model I'. This stable model I' has a corresponding causal stable model I, such that $I' = I^{cl}$ and both interpretations coincide in their assignment of 0's. Therefore, $P^I = P^{I'}$ and we can use the latter to iterate the T_P operator and obtain the least causal model of this reduct, which will mandatorily be a causal stable model due to Theorem 6.

6 Related Work

Cabalar (2011) already introduced the main motivations of our work, but used *ad hoc* operations on proof trees without resorting to algebraic structures. A preliminary version (Cabalar and Fandinno 2013) of the current approach relied on chains of labels but was actually *weaker*, missing basic properties we can derive now from causal graphs.

There exists a vast literature on causal reasoning in Artificial Intelligence. Papers on reasoning about actions and change (Lin 1995; McCain and Turner 1997; Thielscher 1997) have been traditionally focused on using causal inference to solve representational problems (mostly, the frame, ramification and qualification problems) without paying much attention to the derivation of cause-effect relations. Perhaps the most established AI approach for causality is relying on *causal networks* (Pearl 2000; Halpern and Pearl 2005; Halpern 2008). In this approach, it is possible to conclude cause-effect relations like "A has caused B" from the behaviour of structural equations by applying the counterfactual interpretation from Hume (1748): "had A not happened, B would not have happened." As discussed by Hall

(2004), the counterfactual-based definition of causation corresponds to recognising some kind of *dependence* relation in the behaviour of a non-causal system description. As opposed to this, Hall considers a different (and incompatible) definition where causes must be connected to their effects via *sequences of causal intermediates*, something that is closer to our explanations in terms of causal graphs.

Apart from the different AI approaches and attitudes towards causality, from the technical point of view, the current approach can be classified as a *labelled deductive system* (Broda *et al.* 2004). In particular, the work that has had a clearest and most influential relation to the current proposal is the *Logic of Proofs* (**LP**) by Artëmov (2001). We have borrowed from that formalism part of the notation for our causal terms and rule labellings and the fundamental idea of keeping track of justifications by considering rule applications.

Focusing on LP, our work obviously relates to explanations as provided by approaches to debugging in ASP (Gebser *et al.* 2008; Pontelli *et al.* 2009; Schulz *et al.* 2013; Damásio *et al.* 2013). Pereira *et al.* (1991) and Denecker and De Schreye (1993) also define different semantics in terms of justifications, but do not provide calculi for them. In these works, explanations usually contain all possible ways to derive an atom or to prevent its derivation, including paths through negation. This differs from a KR orientation where only the cause-effect relations that "break the norm" should be considered relevant. This point of view is also shared, e.g., by the counterfactual-based causal LP approach (Vennekens 2011). Fages (1991) characterised stable models in terms of loop-free justifications expressed as partial order relations among atoms in positive bodies. We conjecture that the causal values obtained in our semantics formally capture Fages' justifications. A more far-fetched resemblance exists to work on the analysis of tabled Prolog computations. There, the goal is to identify potential causes for non-termination of program evaluations, which can be achieved examining so-called *forest logs*, i.e., a log of table operations for a computation. By adding unique labels for rules (with the original intention to disambiguate analysis results, cf. Liang and Kifer (2013), however not as an explicit means for representing knowledge), in principle a forest log implicitly contains the information necessary to read of the causal model of a completely labelled positive causal logic program.

7 Conclusions

In this paper we have provided a multi-valued semantics for normal logic programs whose truth values form a lattice of causal graphs. A causal graph is nothing else but a graph of rule labels that reflects some order of rule applications. In this way, a model assigns to each true atom a value that contains justifications for its derivation from the existing rules. We have further provided three basic operations on the lattice: an addition, that stands for alternative, independent justifications; a product, that represents joint interaction of causes; and a concatenation that reflects rule application. We have shown that, for positive programs, there exists a least model that coincides with the least fixpoint of a direct consequences operator, analogous to van Emden and Kowalski (1976). With this, we are able to prove a direct correspondence between the semantic values we obtain and the syntactic idea of proof. These results have been extrapolated to stable models of programs with default negation, understanding the latter as "absence of cause." Although, for space reasons, we have not dealt with programs with variables, their semantics is obtained from their (possibly infinite) grounding, as usual.

Several topics remain open for future study. An interesting issue is to replace the syntactic definition by a reduct in favour of a logical treatment of default negation, as has been done for (non-causal) stable models and their characterisation in terms of Equilibrium Logic (Pearce 2006). Regarding the representation of causal information, a natural next step would be the consideration of syntactic operators for more specific knowledge like the influence of a particular event or label in a conclusion, expressing necessary or sufficient causes, or even dealing with counterfactuals. Further ongoing work is focused on implementation, complexity assessment, and an extension to disjunctive programs, respectively the introduction of strong negation. Exploring related areas of KR and reasoning, such as, e.g., Paraconsistent Reasoning and Belief Revision, seems promising with respect to extending the range of problems to which our approach may effectively be applied.

Acknowledgements

We are thankful to David Pearce, Manuel Ojeda, Jesús Medina, Carlos Damasio and Joost Vennekens for their suggestions and comments on earlier versions of this work. We also thank the anonymous reviewers for their help to improve the paper.

Supplementary material

To view supplementary material for this article, please visit http://dx.doi.org/10.1017/S1471068414000234.

References

Artëmov, S. N. 2001. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic 7,* 1, 1–36.

Brewka, G., Eiter, T., and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM 54,* 12, 92–103.

Broda, K., Gabbay, D., Lamb, L., and Russo., A. 2004. *Compiled Labelled Deductive Systems: A Uniform Presentation of Non-Classical Logics.* Research Studies Press.

Cabalar, P. 2011. Logic programs and causal proofs. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning.* AAAI.

Cabalar, P. and Fandinno, J. 2013. An algebra of causal chains. In *Proc. of the 6th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'13).*

Damásio, C. V., Analyti, A., and Antoniou, G. 2013. Justifications for logic programming. In *Proc. of the 12th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning, (LPNMR'13).* Lecture Notes in Computer Science, vol. 8148. Springer, 530–542.

Denecker, M. and De Schreye, D. 1993. Justification semantics: A unifiying framework for the semantics of logic programs. In *Proc. of the Logic Programming and Nonmonotonic Reasoning Workshop.* 365–379.

Fages, F. 1991. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. *New Generation Computing 9,* 3–4, 425–443.

Gebser, M., Pührer, J., Schaub, T., and Tompits, H. 2008. Meta-programming technique for debugging answer-set programs. In *Proc. of the 23rd Conf. on Artificial Inteligence (AAAI'08).* 448–453.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming: Proc. of the Fifth International Conference and Symposium (Volume 2),* R. A. Kowalski and K. A. Bowen, Eds. MIT Press, Cambridge, MA, 1070–1080.

Hall, N. 2004. *Two concepts of causality*. 181–276.

Halpern, J. Y. 2008. Defaults and normality in causal structures. In *Proc. of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008).* 198–208.

Halpern, J. Y. and Pearl, J. 2005. Causes and explanations: A structural-model approach. part I: Causes. *British Journal for Philosophy of Science 56,* 4, 843–887.

Hitchcock, C. and Knobe, J. 2009. Cause and norm. *Journal of Philosophy 11*, 587–612.

Hume, D. 1748. An enquiry concerning human understanding. Reprinted by Open Court Press, LaSalle, IL, 1958.

Kifer, M. and Subrahmanian, V. S. 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming 12.*

Liang, S. and Kifer, M. 2013. A practical analysis of non-termination in large logic programs. *TPLP 13,* 4–5, 705–719.

Lin, F. 1995. Embracing causality in specifying the indirect effects of actions. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, C. S. Mellish, Ed. Morgan Kaufmann, Montreal, Canada.

McCain, N. and Turner, H. 1997. Causal theories of action and change. In *Proc. of the AAAI-97*. 460–465.

McCarthy, J. 1977. Epistemological problems of Artificial Intelligence. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI).* MIT Press, Cambridge, MA, 1038–1044.

McCarthy, J. 1998. Elaboration tolerance. In *Proc. of the 4th Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense 98).* London, UK, 198–217. Updated version at
`http://www-formal.stanford.edu/jmc/elaboration.ps`.

Pearce, D. 2006. Equilibrium logic. *Ann. Math. Artif. Intell. 47,* 1-2, 3–41.

Pearl, J. 2000. *Causality: models, reasoning, and inference.* Cambridge University Press, New York, NY, USA.

Pereira, L. M., Aparício, J. N., and Alferes, J. J. 1991. Derivation procedures for extended stable models. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, J. Mylopoulos and R. Reiter, Eds. Morgan Kaufmann, 863–869.

Pontelli, E., Son, T. C., and El-Khatib, O. 2009. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming 9,* 1, 1–56.

Schulz, C., Sergot, M., and Toni, F. 2013. Argumentation-based answer set justification. In *Proc. of the 11th Intl. Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'13).*

Stumme, G. 1997. Free distributive completions of partial complete lattices. *Order 14*, 179–189.

Thielscher, M. 1997. Ramification and causality. *Artificial Intelligence Journal 1–2,* 89, 317–364.

van Emden, M. H. and Kowalski, R. A. 1976. The semantics of predicate logic as a programming language. *J. ACM 23,* 4, 733–742.

Vennekens, J. 2011. Actual causation in cp-logic. *TPLP 11,* 4–5, 647–662.